AFRL-RI-RS-TR-2017-036

DARPA ROBOTICS CHALLENGE (DRC) USING HUMAN-MACHINE TEAMWORK TO PERFORM DISASTER RESPONSE WITH A HUMANOID ROBOT

FLORIDA INSTITUTE FOR HUMAN AND MACHINE COGNITION INC.

FEBRUARY 2017

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2017-036 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
ROGER J. DZIEGIEL, JR
Work Unit Manager

/ S /
MICHAEL J. WESSING
Deputy Chief, Information Intelligence
Systems and Analysis Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
FEBRUARY 2017	FINAL TECHNICAL REPORT	SEP 2012 – AUG 2016

4. TITLE AND SUBTITLE

DARPA ROBOTICS CHALLENGE (DRC)
USING HUMAN-MACHINE TEAMWORK TO PERFORM DISASTER RESPONSE WITH A HUMANOID ROBOT

5a. CONTRACT NUMBER
FA8750-12-1-0316

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62702E

6. AUTHOR(S)

Jerry Pratt

5d. PROJECT NUMBER
ROBO

5e. TASK NUMBER
PR

5f. WORK UNIT NUMBER
07

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Florida Institute for Human and Machine Cognition Inc.
40 South Alcaniz Street
Pensacola, Fl 32502-6008

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RIED
525 Brooks Road
Rome NY 13441-4505

DARPA
675 North Randolph St
Arlington, VA 22203-2114

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2017-036

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report presents an overview of the work done by Team Institute for Human and Machine Cognition (IHMC) from 2012-2016 through three phases of the Defense Advanced Research Projects Agency (DARPA) Robotics Challenge (Virtual Robotics Challenge (VRC), Trials, and Finals), as well as an extended research phase. It gives an overview of their general workflow for producing state-of-the-art robotics software, as well as focusing on the challenges and techniques used to move beyond the environments and tasks encountered in the DARPA Robotics Challenge. Namely, increasing walking robustness and task autonomy in the Atlas robot. This report is broken down into the four phases of IHMC's participation in the DARPA Robotics Challenge: Phase 1, Virtual Robotics Challenge; Phase 2, Trials; Phase 3, Finals; and Phase 4, Extended Research. As competitor in the DARPA Robotics Challenge (DRC) the IHMC team was able to achieve second place in the final stage of the competition held in the summer of 2015. Previous to that the team was awarded first and second place in the two earlier stages of the competition: the VRC, and the DRC Trials.

15. SUBJECT TERMS

Robotics, Mobility, Platform Dexterity, Supervised Autonomy, Wireless, Ground

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**ROGER J. DZIEGIEL, JR**
U	U	U	UU	73	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

Table of Contents

List of Figures iii

1.0 EXECUTIVE SUMMARY 1

2.0 INTRODUCTION 2

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES 2

3.1 PHASE 1: VIRTUAL ROBOTICS CHALLENGE 2

3.1.1 Demo0 3
3.1.2 Demo1 4
3.1.3 Stair Climbing 5
3.1.4 Software Infrastructure 6
3.1.5 Java Monkey Engine toolbox for 3D Graphics 7
3.1.6 Multi-contact Capturability Theory and Analysis Tools 7
3.1.7 Whole-body Control Framework 8
3.1.8 High Level Control Behaviors 9
3.1.9 Swing Leg Trajectory Generator 10
3.1.10 ROS and Gazebo Integration 11
3.1.11 Running on the Constellation 13
3.1.12 Development Tools 13
3.1.13 Communication Protocol 14
3.1.14 Autonomous Driving 16
3.1.15 Emergency Recovery Behaviors 16
3.1.16 Operator Interface 17
3.1.17 Walking Task 18
3.1.18 Hose Task 19
3.1.19 Driving Task 20
3.1.20 Scripting 21
3.1.21 Operator Training 22
3.1.22 Virtual Robotics Challenge Results 26

3.2 Phase 2: DARPA Robotics Challenge – Trials 28

3.2.1 Sensing 29
3.2.2 Sensor Synchronization 29
3.2.3 Accounting for LIDAR Rotation 29
3.2.4 LIDAR Shadows 30
3.2.5 Joint angle offsets and calibration 30
3.2.6 Arm Calibration 31
3.2.7 Walking 32
3.2.8 Hands 33
3.2.9 User Interface 35
3.2.10 Coloring and texture 38
3.2.11 A wider field of view 38
3.2.12 Understanding joint limits 39
3.2.13 Infrastructure 39
3.2.14 Developer Infrastructure 41

3.2.15 Gazebo 42
3.2.16 Qualifying 42
3.2.17 Trials Emulation 43
3.2.18 Practicing Competition Tasks 44
3.2.19 Trials Competition in Miami, Florida 45
3.3 Phase 3: DARPA Robotics Challenge – Finals 46
3.3.1 Car Driving 49
3.3.2 Car Egress 50
3.3.3 Practice, Practice, Practice 51
3.3.4 Robot Operation 51
3.4 Phase 4: Extended Research 52
3.4.1 Logging 52
3.4.2 Testing and Debugging 52
3.4.3 Running on Hardware 53
3.4.4 Enhancing the IHMC Control Framework 53
3.4.5 Walking Partial Footholds 55
3.4.6 Autonomy of High Level Behaviors While Retaining Coactive Design Methodology 56
4.0 RESULTS AND DISCUSSION 58
4.1 The VRC: Results 58
4.2 The Trials: Results 58
4.3 The DRC Finals: Results 59
4.4 Extended research: Results 60
5.0 CONCLUSIONS 61
6.0 PUBLICATIONS AND PRESENTATIONS 62
6.1 Publications 62
6.2 Presentations 63
7.0 LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS 64

List of Figures

Figure 1. Executing Demo0. Left: box to lift (red) and location to put down (blue) have been selected through the GUI. Middle: robot crouches, leans and rotates to pick up box from low table. Right: robot puts box down on high table. 4
Figure 2. GFE robot model as rendered in our simulation software. 5
Figure 3. Simulated LIDAR in a simple test world, yet to be integrated into the main simulation and the user facing interface. All points at maximum range are displayed, though they will be culled before being shown to the user. The scene includes a Utah teapot and a cube. 5
Figure 4. Left: robot climbing steep stairs. Right: varying height stairs. 6
Figure 5. A high level behavior, such as a walking or driving behavior, supplies the whole-body control framework with the following data: 1) motion constraints, 2) a rate of change of centroidal momentum objective, and 3) available contact information. 8
Figure 6. Contact points and basis vectors approximating the friction cones for a planar contact (e.g. a foot). 9
Figure 7. Two foot trajectories. The white balls are the waypoints, and the black ones are samples of the trajectory spaced equally in time. 11
Figure 8. DRC Dashboard, visualizing which ‘cloud’ computers are available to run the simulation or control software. Task world selection and starting location selection are available. 14
Figure 9. Network layout for the IHMC VRC entry. The control software communicates with the DRC simulator through a high bandwidth connection at a rate of 1000 Hz. The Operator Control Unit communicates through a network processor at low bandwidth and high latency. 15
Figure 10. Height map of terrain based on LIDAR. 17
Figure 11. Variable Octree resolution for the hose task. 18
Figure 12. Footstep planning in the user interface. Footsteps snap to the LIDAR world map, and are always planned along straight-line paths. The control ring at the end of a footstep path can be dragged to change the target, and can also be used to change the direction in which the robot is facing during locomotion, allowing sidesteps and backwards walking. Footstep colors signify various levels of perceived danger. 19
Figure 13. Hose manipulable. 19
Figure 14. Graphic depiction of valid (left side) green IK solutions and invalid (right side) red IK solutions. 20
Figure 15. Vehicle ingress. 20
Figure 16. Driving interface 21
Figure 17. Snapshot of the user interface where the operator is preparing to command the robot to exit the next desired pelvis pose in a script, as visualized by the semitransparent, yellow 3D model of the pelvis. 21
Figure 18. Script Editor Interface with three total scripts loaded in. 22
Figure 19. Custom driving course made to test our driving controller. Notable obstacles include houses obstructing the view of barrels, barrels obstructing the view of the sides of the road, and a “drive-thru” gas station. 23

Figure 20. Custom walking course with steeper hills and a narrow, cinder block-laden canyon. 24
Figure 21. Custom manipulation task with a larger and more massive hose connector, lower table, valve and standpipe placement reversed, and smaller valve with higher damping parameter. 24
Figure 22. Custom walking task consisting of four mud pits with increasing friction and damping parameters. 25
Figure 23. Custom world with heightmap made from the face of our Principle Investigator/fearless leader. 25
Figure 24. System Architecture. 29
Figure 25. Left: Camera image of valve. Right: Point cloud showing LIDAR shadow. Note how edges of the valve have false points going back to the wall. The rainbow color depth map provides a mechanism for the operator to estimate the actual thickness of an object. 30
Figure 26. Arm calibration routine showing the residual error after bias estimation. Red dots are the calibration points on the target. Blue dots are the predicted location given our kinematics model and the bias estimate from batch optimization. The gray dot is the end effector position before any correction and the yellow dot is the change after correction. 31
Figure 27. Hook hand and iRobot hand used in combination. 33
Figure 28. Modified routing hole of iRobot hand for reduced tendon breakage. 34
Figure 29. Atlas with 8-inch arm extensions during the debris task. 34
Figure 30. Our UI with the live camera image (left side of UI) wrapped by the fisheye image (too dark to tell in this image). The lidar can be displayed as a quadtree, point cloud or a colorized point cloud (shown on the ride side of UI). 36
Figure 31. Door interactive object in our UI. 37
Figure 32. Image on the left represents the actual camera image as seen in the user interface. The center image shows the texturing feature on the point cloud data. The right image shows the coloring feature on the point cloud data. 38
Figure 33. Images from head and chest cameras fused together into the first person viewport. The higher resolution rectangle is the normal camera and the rest of the image is from the fisheye camera. This gives the operator the illusion of looking to the right even though the robot cannot yaw its head. 39
Figure 34. Color indicators used to show that joints are reaching their physical limits. . 39
Figure 35. Logging computer showing recorded video synchronized with SCS playback of data and graphics. 40
Figure 36. Bamboo Wall Board displaying the status of our suite of test cases. 41
Figure 37. Atlas with iRobot hands in Gazebo. Notice how the fingertips are turned the wrong way due to incorrect modeling of the proximal finger joints. 42
Figure 38. Task emulations at the IHMC Robot Lab. 44
Figure 39. TEAM IHMC at the DRC Trials in Miami, Florida. 45
Figure 40. Team IHMC's Atlas performing the task at the DRC Trials. 45
Figure 41. DRC Finals Team Standings. Team IHMC completed all 8 tasks in 50 minutes and 26 seconds, which was good for a second-place finish and a $1M prize from DARPA. 46

Figure 42. Team IHMC fell two times during the first day of the DRC finals. The fall on cinder blocks was due to taking an aggressive step without selecting a walking configuration used for aggressive steps. The fall on the stairs was due to a combination of rushing since we were low on time, a programming bug, and body oscillations that we think were a result of falling on the cinder blocks. After day one, we made corrections for both falls and were able to successfully complete all eight tasks on day two. 47
Figure 43. Team IHMC Running Man completing the 8 tasks, plus celebrating at the end, at the DARPA Robotics Challenge Finals. 48
Figure 44. One of our goals for the DRC was to engage the crowd and allow visitors to see the robot and operator interface. We had three IHMC “Blue Men” release nearly 200 T-shirts into the crowd during our two runs. In our garage, we gave demonstrations to over 100 people and interviews with over 10 press organizations. 49
Figure 45. Left: picture of the vehicle modification for steering. Right: Image of vehicle modification for pressing the gas pedal. 49
Figure 46. Picture of the vehicle with the checkerboard sticker mounded on the hood. The checkerboard acted as a fiducial marker to enable automated localization of the robot in the vehicle. 50
Figure 47. Picture of the vehicle with the step addition. The step provided an intermediate platform so that the robot did not have to step down the full vehicle height in a single step. 50
Figure 48. Operator UI (left), Primary operator (John Carff), Strategist/Planner (Matt Johnson), Co-Pilot (Duncan Calvert), Strategist/Planner/Field Team Communicator (Jerry Pratt’s arm in top right). 51
Figure 49. Procedure chosen to improve the controller reliability. 53
Figure 50. The simulated Atlas robot walking over randomly oriented, line shaped stepping stones and a point foothold. After each step the new foothold is explored and the control algorithm adjusts the stepping accordingly. 55
Figure 51. The Atlas humanoid walking over variously oriented line contacts. We use tilted cinder blocks to produce line contacts. It can be seen that the feet of the robot stay flat and do not conform to the cinder block surfaces. 55
Figure 52. Atlas autonomously picking balls off of the floor using a high level behavior. While this is a very simple behavior, it demonstrates a fully autonomous task that integrates sensing, perception, whole body mobility, and simple manipulation. 57
Figure 53. Behavior flow chart for autonomously picking up balls with Atlas. 57

1.0 EXECUTIVE SUMMARY

Title: DARPA Robotics Challenge - Team IHMC Final Technical Report

Principal Investigator: Jerry Pratt

Research Objectives for this final Reporting Period:

- Increase the amount of autonomy in humanoid behaviors, while allowing for coactive design of user interfaces.
- Walk on surfaces with only partial footholds, such as on the tops of pointy rocks
- Use upper body angular momentum to help balance.
- Improve the software architecture and interface to the whole-body momentum based control framework.

Synopsis of Research:

This report presents an overview of the work done by Team Institute for Human and Machine Cognition (IHMC) from 2012-2016 through three phases of the Defense Advanced Research Projects Agency (DARPA) Robotics Challenge (Virtual Robotics Challenge (VRC), Trials, and Finals), as well as an extended research phase. It gives an overview of our general workflow for producing state-of-the-art robotics software, as well as focusing on the challenges and techniques used to move beyond the environments and tasks encountered in the DARPA Robotics Challenge. Namely, increasing walking robustness and task autonomy in the Atlas robot. This report is broken down into the four phases of IHMC's participation in the DARPA Robotics Challenge: Phase 1, Virtual Robotics Challenge; Phase 2, Trials; Phase 3, Finals; and Phase 4, Extended Research.

2.0 INTRODUCTION

As competitor in the DARPA Robotics Challenge (DRC) the IHMC team was able to achieve second place in the final stage of the competition held in the summer of 2015. Previous to that the team was awarded first and second place in the two earlier stages of the competition: the VRC, and the DRC Trials.

This success was made possible by the research described in this report. It highlights that the work done at IHMC before and during the competition is state of the art in control of legged robots as well as in human-robot cooperation. The balancing algorithms, the manipulation control, and the operator interfaces developed by the teams are only the obvious accomplishments of the DRC. When it comes to deploying real robots in real world scenarios difficulties arise that cannot fully be anticipated beforehand. In the following we will give an overview of all these aspects and describe how our team was able to overcome difficulties along the way.

The DRC was designed to advance humanoid robots to a point where they could be used in disaster scenarios like the nuclear power plant disaster in Fukushima. They should be able to go to places that humans can not go to because of radiation or other dangers. There they should do some basic manipulation tasks like closing valves or flipping switches. In the first stage of the competition, the VRC, teams developed the tools needed to achieve these goals in software simulation. In the second stage of the competition, the Trials, those algorithms had to be moved to a real robot, in the case of IHMC the Atlas robot. Finally, in the last stage the robots needed to demonstrate their capabilities during a simulated one hour disaster relieve mission. In the following we will describe the teams profess, the challenges, and the achievements during each of these stages in one chapter.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 PHASE 1: VIRTUAL ROBOTICS CHALLENGE

During the Virtual Robotics Challenge, which was held June 18–20 2013, teams attempted to complete three of the DRC tasks in a computer simulation using a simulated version of the robot ‘Atlas’, made by Boston Dynamics (BDI). The three VRC tasks were to have the robot:

1) climb into a Polaris Ranger vehicle, drive along a road with obstacles, exit the vehicle and walk across finish;
2) walk across various rough terrain: a mud pit, rolling hills, and a debris field;
3) attach a hose to a spigot and subsequently turn a valve to start fluid flow.

Team IHMC completed all of these tasks multiple times in competition, and scored a total of 52 out of a possible 60 points related to task completion, placing first in a field of 22 teams. Our preparations for the VRC include: a new whole body control algorithm, a novel

modular state estimator, a coactive-design-inspired operator user interface, and a low-bandwidth communication infrastructure.

Prior to announcement of the Virtual Robotics Challenge (VRC), we focused on developing core software necessary for manipulation. After the VRC was announced, we decided that our high-level strategy would be tackling the walking problem first. This decision was made based on our team's strengths in walking algorithms, our extensive pre-existing software tools available to sufficiently simulate and analyze the walking problem, and the insufficient state of the VRC simulator. We planned to leverage our tools and skills to develop a system in which we can get the simulated government furnished equipment (GFE) robot to walk over various types of terrain, including but not limited to: flat, slopes, steps, rocks, and gaps. We planned to work on terrain first, in which hands are not required, and then work on terrain requiring handholds.

All of the internal demos we have done so far are built upon a common control framework, which is based on Instantaneous Capture Point (ICP) dynamics and the rate of change of momentum. This control framework, as well as the behaviors built on top of it, can easily be ported between different robot morphologies.

We planned to shift focus to either the manipulation challenge or the driving challenge as soon as our performance in the walking challenge was adequate.

3.1.1 Demo0

Prior to the VRC announcement, we started out with a simple balance and manipulation demo: move boxes between tables in front of the robot. This challenge allowed us to extend our expertise into manipulation and develop some user interface tools for manipulation.

The goal of Demo0 was to demonstrate our ability to do simple force controlled manipulation in the absence of locomotion. For this purpose, we created a simulated world in which there are three tables of different heights and two boxes. We then had our 30 degree of freedom simulated robot move the boxes between the tables (see Fig. 1). The operator can select a box, as well as where to put it, by clicking in the simulated world. When selecting a box, the operator can specifically select the faces of the box which the robot will use to pick it up. Therefore, depending on the robot's orientation with respect to the box, the operator can easily select the best two faces to securely interact with the box. Commands to pick up the selected box and to put it down can also be given through the Graphical User Interface (GUI). Once a box has been selected for pick-up the operator then chooses the target destination by moving around a semi-transparent blue box of the same size and shape to the desired location.

The robot performs position control on the box in Cartesian space, both with respect to the world and with respect to its own chest, while also controlling the compression forces exerted on the box by the hands. While performing these tasks, the robot also controls its posture. The robot can lean towards a box, rotate its pelvis and chest independently, as

well as control its center of mass height from anywhere between a full crouch to an upright stance.

During Demo0, the robot's fingers were locked in fixed positions. The problem of grasp control was deferred to a later demo. The robot morphology was close to, but not the same as the GFE robot.

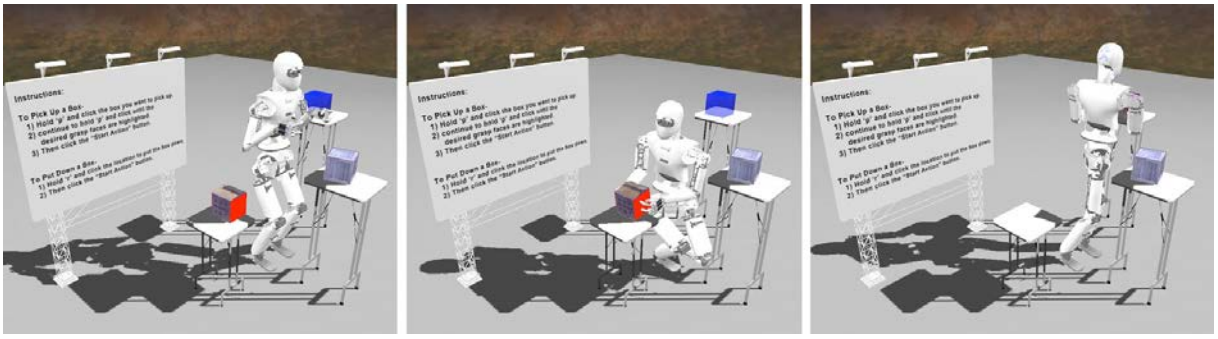

Figure 1. ***Executing Demo0. Left: box to lift (red) and location to put down (blue) have been selected through the GUI. Middle: robot crouches, leans and rotates to pick up box from low table. Right: robot puts box down on high table.***

3.1.2 Demo1

Demo1 (see Fig. 2) directly addresses the walking task challenge and focuses on navigating through an environment with large obstacles that need to be avoided. We are starting with flat ground and will include rougher terrain in subsequent demos.

Demo1 required the development of the operator user interface and the network communications between the simulation and this user interface. Whereas in Demo0 the operator could give the robot commands by interacting directly with the simulation environment, in Demo1 the operator is only be able to interact with the robot via a remote user interface. Data from the simulation is streaming to the interface over Transmission Control Protocol (TCP). The data includes a video from the onboard camera and joint position information. This data drives the user interface (UI).

We have switched from using the robot model employed in Demo0 to the actual GFE robot model. We have written software that enables us to import Simulation Description Files (SDF), used by Gazebo, into our own simulation and control software. Using the SDF description file of the GFE robot allows us to quickly incorporate changes in our software. During the development of the SDF import library, we found and reported several errors in the GFE definition. Due to the nature of our control algorithms, switching between robot models involves few parameter changes.

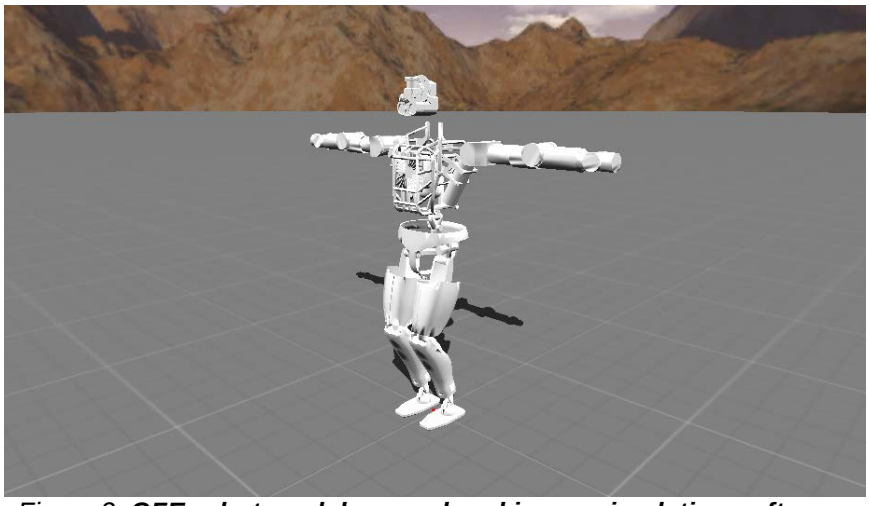

Figure 2. ***GFE robot model as rendered in our simulation software.***

Through the remote user interface, an operator is able to create virtual objects that represent obstacles, create straight line paths, plan footsteps based on this path and command execution of the specified footsteps. These footsteps are then sent to the robot controller in the simulation over TCP. The controller then executes the planned footsteps until the destination is reached, and comes to a stop. We are also working on implementing a simulated Light Intensity Distance and Ranging (LIDAR) sensor, and incorporating it into the operator GUI and the footstep planning algorithms. Our simulated LIDAR has been successfully integrated into the same graphics framework we use for our simulation, but has yet to be added to the robot.

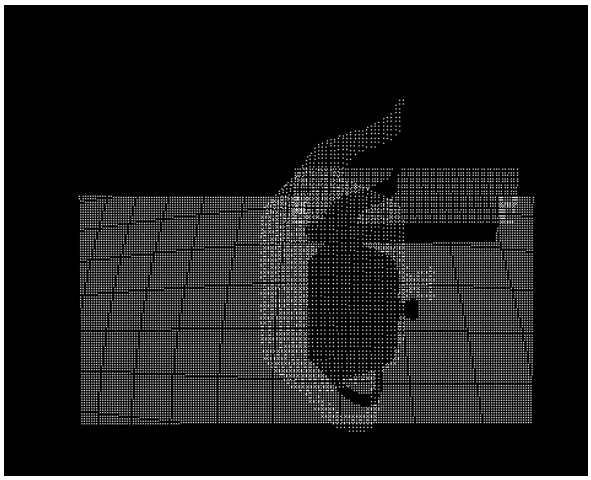

Figure 3. ***Simulated LIDAR in a simple test world, yet to be integrated into the main simulation and the user facing interface. All points at maximum range are displayed, though they will be culled before being shown to the user. The scene includes a Utah teapot and a cube.***

3.1.3 Stair Climbing

In parallel to Demo0 and Demo1, we have extended our walking algorithms to varying height terrain, for the DRC as well as for other projects. In particular, we have focused on

climbing a steep set of stairs. We are currently able to make our simulated robot climb stairs with a rise of 35 cm and a tread of only 15 cm at a fairly quick rate of 7.9 seconds per ten steps (see Fig. 4). Stairs this steep disallow the use of the full foot polygon to balance on a step by Center of Pressure (CoP) placement, forcing us to focus on the use of angular momentum and improved weight distribution algorithms for balance control. In addition, combined ground reaction force control and foot orientation control is necessary, where the orientation of the foot is constrained by the line contact with the step. We are also able to climb variable height stairs.

Stair climbing fits nicely into our momentum and ICP based control framework. In addition, almost the same high level behavior is used for flat ground walking and stair climbing, with only minor modifications to e.g. Center of Mass (CoM) height trajectories. We plan to make full use of the algorithms developed for varying height stairs in the DRC, enabling dynamic, extremely rough terrain locomotion.

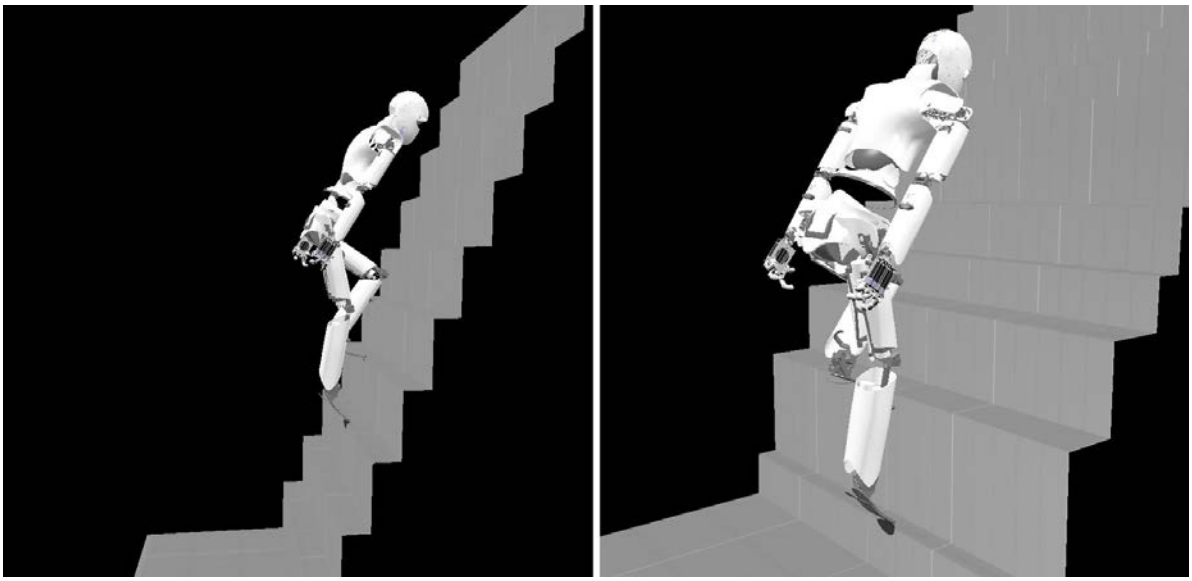

Figure 4. ***Left: robot climbing steep stairs. Right: varying height stairs.***

3.1.4 Software Infrastructure

One of the main challenges of complex robotics projects is managing code complexity. Therefore, we put a great deal of effort into software infrastructure and software engineering best practices. Our team workflow leverages the principles of Continuous Integration to help manage and maintain code quality; for each software module that we developed, we wrote small unit tests that verify the functionality of the software. For each of the demos we developed, we created an automatic demo runner that executes the demo, simulating a simple sequence of actions that the operator might take in interacting with the demo and asserting that performance has not degraded. All of our software is continuously verified with an automatic build server that executes our test cases at each code commit from any developer. We also run longer, more fine-grained nightly builds and tests every evening.

One major infrastructure area we dedicated effort to this quarter was in making sure all of our algorithms are rewindable, which allows for much better debugging. Instead of having to rerun a simulation from start when it fails, the user can rewind it a few seconds, change parameters, and resimulate from where that point in time on. In order to keep the simulations rewindable however, one must use discipline and make sure that every variable used in a control algorithm that represents internal state be specified. In addition to making sure our simulations were rewindable, we also developed new testing software for helping to identify where in the code the culprit lies when an algorithm is not rewindable.

3.1.5 Java Monkey Engine toolbox for 3D Graphics

Up to this point, our simulation and analysis software tools have used Java3D for 3D graphics. During this quarter, we have switched much of our software over to using Java Monkey Engine (JME) instead. JME has better features than Java3D and is more actively developed. It also includes software for collision modeling, and has a dynamics engine, based on Bullet, which may also be the engine used for the VRC.

In order to switch over to JME without disrupting other work, we first developed a Graphics3DAdapter interface, which contains the various methods and classes that we would like to use for interacting with any 3D Graphics environment. We then refactored our code so that our Java3D-based software implements the Graphics3DAdapter interface. Next we implemented the interface using JME. In this way, we can now interchangeably switch between Java3D and JME. However, as we move forward, we will likely just use JME directly since the features we will be using may not all be implemented in Java3D. Switching over to JME has also allowed us to create a more efficient simulated LIDAR sensor using the 3D engine's ray tracing functionality.

3.1.6 Multi-contact Capturability Theory and Analysis Tools

For some rough terrain walking we think it will be important to use hands to help balance. Therefore, we have been developing the theory and analysis tools for multi-contact capturability. Each surface is defined by a surface normal and friction cone. There are several questions we are interested in. The first, a question of static equilibrium, can be stated as: Given the contact conditions of the robot, what are the states of the robot that can be held in static equilibrium, i.e. for what positions can the velocities and accelerations be zero? This is essentially a solved problem with several very good papers that we identified and implemented the algorithms for. A more challenging problem is zero-step capturability, which can be stated as: Given the contact conditions of the robot, can the robot prevent falling down without taking a new step or grabbing a new handhold. For this problem, we identified one potential algorithm, implemented portions of it, and are currently refining the algorithm. Once that is finished, we will investigate the yet harder problem of one-step capturability, which can be stated as: Given the contact conditions and state of the robot, can the robot prevent falling down by taking one additional step, or grabbing one additional handhold.

3.1.7 Whole-body Control Framework

During this phase, we finished designing and implementing our whole-body motion control framework. Figure 5 shows the flow of information in this framework. It is heavily influenced by recent work from Orin and Goswami and colleagues and also incorporates features from work by Sentis and colleagues.

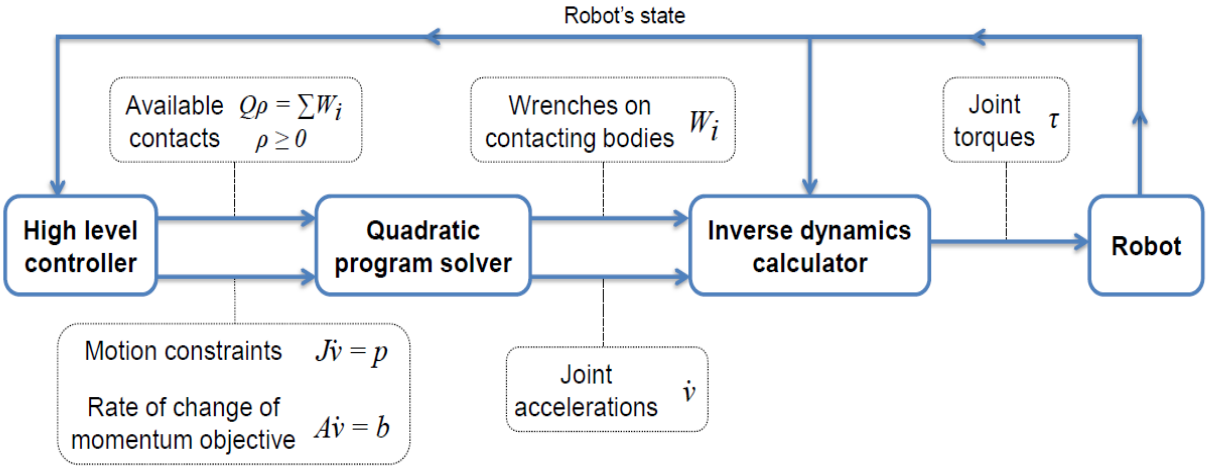

Figure 5. ***A high level behavior, such as a walking or driving behavior, supplies the whole-body control framework with the following data: 1) motion constraints, 2) a rate of change of centroidal momentum objective, and 3) available contact information.***

A motion constraint with index i can be written as

$$J_i \dot{v} = p_i$$

where $\dot{v} \in \mathbb{R}^{n+6}$ is the vector of joint accelerations, including the spatial acceleration of the pelvis with respect to the world. Here, *x* is the number of actuated degrees of freedom. We mainly use three types of motion constraints: joint space acceleration constraints, for example specifying the rotational acceleration of an elbow joint; spatial acceleration constraints between rigid bodies or with respect to the world, for example specifying the linear and angular acceleration of a frame attached to the wrist with respect to the chest; point acceleration constraints, specifying a desired acceleration of a body fixed point with respect to the world, for example specifying the desired linear acceleration of a point on a foot with respect to the world. The individual motion constraints are then concatenated into a single matrix equation

$$J\dot{v} = p$$

In addition to these motion constraints, a rate of change of momentum objective can be provided by the high-level behavior in the form

$$A\dot{v} = b$$

where *A* is the centroidal momentum matrix and *b* encapsulates the desired rate of change of centroidal momentum (both linear and angular).

Contact information is supplied in the form of a contact matrix *Q*, which is built from the available points of contact (e.g. foot corner points, or contact points on the robot's thighs)

and a set of basis vectors, (Fig. 6). The basis vectors form a conservative pyramid approximation of the friction cone at each contact point, so that a nonnegative linear combination of these basis vectors results in a contact point force that is within the friction cone. For each contacting body, the component of the body wrench from is a linear combination of the basis vector multipliers. Inequality constraints ensure the unilaterality of the contact forces. Summing all of the contact wrenches, we arrive at

$$\sum_{i=1}^{m} W_i = Q\rho, \quad \rho \geq 0$$

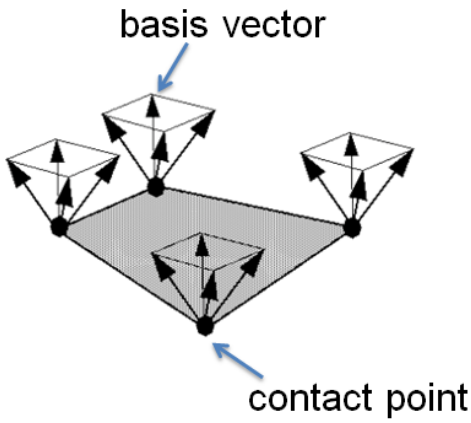

Figure 6. ***Contact points and basis vectors approximating the friction cones for a planar contact (e.g. a foot).***

Combining all of the objectives and constraints, we formulate the following quadratic program:

$$\begin{aligned} \text{minimize} \quad & (A\dot{v} - b)^T W_h (A\dot{v} - b) + \rho^T W_\rho \rho + \dot{v}^T W_{\dot{v}} \dot{v} \\ & Q\rho = A\dot{v} + \dot{A}v - W_g \\ \text{subject to} \quad & \rho \geq 0 \\ & J\dot{v} = p \end{aligned}$$

where the *W* weighting matrices in the objective function are determined by the high level controller. This quadratic program is solved every controller time step for the joint acceleration vector and the ground reaction force basis vector. This information is then used to calculate a vector of desired joint torques using a recursive Newton-Euler inverse dynamics algorithm.

The advantage of this joint optimization of contact forces and joint accelerations is that it ensures that any planned robot motions will always be feasible, given the available contacts and their friction cones.

3.1.8 High Level Control Behaviors

We developed several high-level behaviors on top of the whole-body control framework. These include:

1. *Direct multi-contact behavior:* The most straightforward application of the whole-body control framework is the direct multi-contact behavior. This high-level behavior allows the operator to directly control the main parts of the robot, namely the feet, hands, pelvis, and chest, by sending desired poses or orientations and commanding whether or not certain contact points or grasps should be used to bear the robot's weight. Minimum jerk trajectories were used to smoothly transition between subsequent poses or orientations. Proportional Derivative (PD) controllers with added feed forward terms were used to compute desired spatial accelerations used as motion constraints in the quadratic program solver. This behavior was chiefly used to perform vehicle ingress and egress in the driving task.
2. *Walking behavior:* As opposed to the direct multicontact behavior, the walking behavior does not specify a motion constraint that controls the pelvis position. Instead, we supply the optimizer with a desired rate of change of whole-body linear momentum. The desired rate of change of linear momentum is computed by a CoM height PD controller for the vertical axis. An ICP controller was used for the horizontal plane. ICP control can be used to compute a rate of change of linear momentum that is likely to be feasible, while enabling indirect control over the robot's CoM position due to the fact that the CoM asymptotically tracks the ICP. A smooth desired ICP trajectory is generated, which corresponds to a constant Centroidal Moment Pivot (CMP) at the stance foot center during single support and a smooth transition of the CMP from previous to upcoming foot center during double support. This avoids jumps in the desired ground reaction forces. We consider the upcoming 3 footstep locations in the design of the desired ICP trajectories.
3. *Driving behavior*: Due to instability in Gazebo's contact solution when the robot is seated, we chose to have the pelvis hover over the seat without contacting it, using only the left hand and left foot for balance. The right hand is used for steering: the driving behavior specifies a point acceleration motion constraint, which controls a certain hand-fixed point to move along an arc while holding the steering wheel, resulting in steering wheel angle control. No orientation control is done for the steering hand, since arm joint limits and kinematics preclude the achievement of arbitrary hand orientations while steering. The right foot is used to press the gas and brake pedals. This is achieved using a point acceleration motion constraint for the robot's toe, and soft gain foot orientation control. Soft gains are used because arbitrary foot orientations are not always achievable due to leg joint position limits.

3.1.9 Swing Leg Trajectory Generator

We developed a general trajectory generator, with the goal of using it for the ankle trajectories, which takes in initial and final positions and velocities and two intermediate waypoints and computes a three-dimensional trajectory as a function of time. See Figure 7 for example ankle trajectories. We experimented with various piecewise schemes with

the goal of creating a trajectory that both had low jerk and a smooth path given any boundary conditions. After finding the piece-wise setup, we made a visualizer and checked its path for a wide range of boundary conditions.

Once we had this tool in place for the ankles, we incorporated a few stepping behaviors into the controller which ultimately fed in the two waypoints to the trajectory generator. These were normal steps, stepping over a box, stepping to avoid toe/heel stubbing, and variable step height. These were incorporated into the UI in various ways: normal steps and stepping to avoid stubbing were walking modes, variable step height needed a height as input, and stepping over a box required the user to create a box and final desired footsteps in the virtual world.

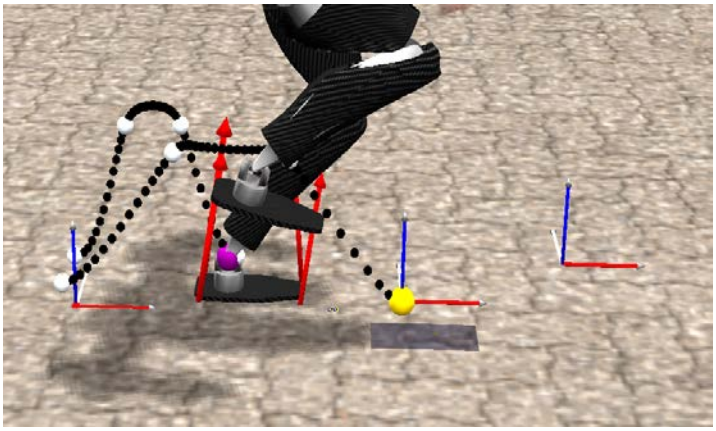

Figure 7. ***Two foot trajectories. The white balls are the waypoints, and the black ones are samples of the trajectory spaced equally in time.***

3.1.10 ROS and Gazebo Integration

For the VRC, it was important to integrate a teams software with Robot Operating System (ROS) and Gazebo. This task was prioritized relatively early to ensure our software development was not constrained to work solely in our own simulator. Two people were picked to learn everything ROS or Gazebo related and to bridge the gap between that environment and our Java code. We elected early on to integrate our pre-existing Java codebase with ROS and Gazebo instead of switching our development to C/C++ as this would allows us to leverage our already substantial walking controls software as well as allowing us to continue to reap the benefits of the Java programming language such as the rich suite of editing and productivity tools as well as the large community of developers and available software libraries. This dictated early on that, as we decided to introduce a custom networking layer on top of the pre-existing ROS stack, we would need to devote a reasonable amount of time to fault tolerance and robustness in the connectivity layer, heavily informing many of our design decisions.

Developing and maintaining instructions for bootstrapping a new computer into a ROS/Gazebo workstation was the first step of our integration process. Creating a set of Bash scripts to automatically download and install the necessary packages and setup

environment variables gave us a way to quickly produce Ubuntu workstations that were identical regarding software, while still leaving the option to tweak individual setups for specific purposes (e.g. driving, walking, manipulation) that is not available when cloning new machines from a system image. The bootstrapping scripts were maintained on our version control server so the latest versions could easily be checked out on new computers to get them up and running. We made the choice early on to use software built from source so we could stay up to date with nightly builds and in turn be plugged in with Open Source Robotics Foundation's (OSRF) debugging and development process; as adherents to the Agile development doctrine and as big fans of Continuous Integration, we knew that performing end-to-end testing of all software stacks would help us mitigate the number of hiccups and gotchas that we encountered on competition day. By investing time into automating our ability to stay on the bleeding edge, we were able to stay fairly in tune with what was and was not working with Gazebo and where our efforts would be best spent in regards to our own software development.

When we started working on Gazebo integration, the ROS layer was not yet in a state compatible with our controller's requirements. Therefore, we created our own plugin that communicates with our control software over TCP. For development purposes, the plugin still has several advantages over the official ROS layer provided by OSRF. Extra features include the ability to get the full robot state, allowing us to verify our state estimator and rudimentary support for rewinding the simulation.

When developing new code, our strategy was to thoroughly test it first in our own simulation software, and then to run it in Gazebo, making whatever tweaks were needed so that it worked properly. To cut down the time spent tweaking code for Gazebo, it was important that our own simulator contained the same models (i.e. collision, world, and sensor noise) and parameters (friction, damping, etc). This became a two-way street where we would try and mold our simulator to match Gazebo's parameters, as well as communicate to OSRF where we thought Gazebo was lacking and suggest improvements.

We created an SDF importer for our simulator so that we could use the same environment and collision models that were provided for Gazebo. The importer parsed and translated the Gazebo SDF models into a format that could be used by our software, allowing our team to develop in a more familiar simulation environment with the same worlds and objects used in Gazebo. There was also the task of ensuring our sensor noise models matched those in Gazebo. This consisted of either digging through ROS and Gazebo source code to see how noise was added or asking directly on the VRC forums what types of noise models would be used for a given sensor.

During initial testing, we found that various parameters of the model, especially inertia of the smaller pieces and damping parameters were extremely large. We have communicated with OSRF about our concerns about those parameters and were able to come up with a solution that worked for all teams. The inertia parameters were chosen to be physically feasible where possible and increased for simulation stability if necessary.

Damping parameters became user settable in the competition, allowing stable joint position control and fast motions for whole body torque control.

3.1.11 Running on the Constellation

Before the constellation became available, we created an in-house setup mimicking the constellation, having two control computers and a simulation computer, and used that to test our software and network protocols. As soon as the constellation became available we started testing our control software on it. During the first days of testing, we found that Gazebo would crash at about 10 minutes simulation time due to a large memory leak. We provided OSRF with our control software and gave them full access to the constellation so they could pin down and fix this bug. After the practice period was over, we were able to test our software quite extensively during the extended practice period.

We developed scripts that automatically deployed our control software to the constellation and made a small helper program that gave us the possibility to restart our control software should something break, using a minimum amount of bandwidth overhead.

Unfortunately, even with all the extensive testing, we found during the competition that we sometimes did not meet the imposed deadlines due to the Java garbage collector pausing our controller. We were able to optimize our control code so that did not happen anymore.

3.1.12 Development Tools

Our team relies heavily on a suit of development tools provided by Atlassian to manage and organize many aspects of our workflow. Their collaboration software, Confluence, serves as a wiki where team members can create pages and make information available to the rest of the team. This functionality was leveraged heavily as we began integrating with ROS and Gazebo, as we could create a collection of instructions for running simulations or known bugs to look out for that was accessible by everyone.

Atlassian's continuous integration server, Bamboo, allowed us to continually run a suite of tests on code we committed to our version control repository and alerting us when new code caused tests to fail. This also let us create "snapshots" of working builds, so that even during the competition we could commit new code knowing that we could easily revert to an earlier snapshot of our code if something broke.

The majority of our team works in Windows and OS X, and forcing everyone to switch to an operating system that is unfamiliar would have cost too much time. To further facilitate the task of working in Linux, those familiar with Ubuntu and Gazebo developed the DRC Dashboard, a GUI for launching simulations over Secure Shell (SSH) from Windows or OS X on a networked machine (see Fig. 8). The DRC Dashboard allows users to select a task and subtask from dropdown menus, and which network computer they wish to run on from a list of available machines. Computers that are already in use show up in red

and are unavailable to launch new simulations. When the user is done, they can stop their simulation from within the GUI to free up the machine for others. All of this was implemented in Java, making use of the Java Secure Channel (JSch) libraries to connect to our Ubuntu workstations over SSH, where we could run executable launch scripts using shell commands to start ROS and Gazebo.

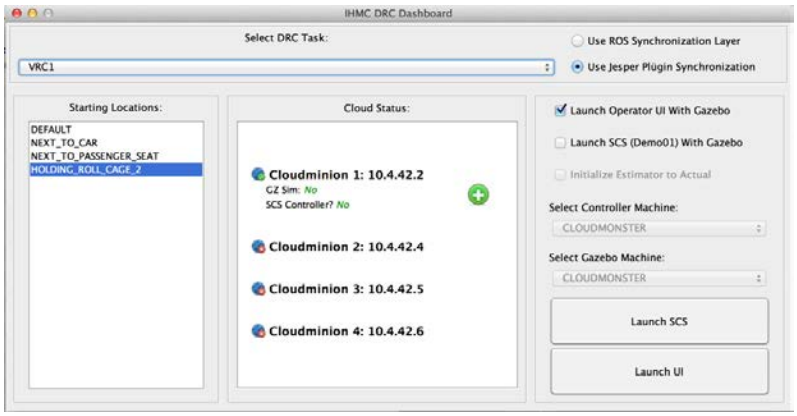

Figure 8. ***DRC Dashboard, visualizing which 'cloud' computers are available to run the simulation or control software. Task world selection and starting location selection are available.***

3.1.13 Communication Protocol

The VRC had very stringent bandwidth requirements, requiring us to come up with a custom communication protocol. After deliberation with DARPA, the bandwidth requirements were redefined as pure data rates, excluding overhead from the ethernet, IPv4 and TCP protocols. At the lowest bandwidth requirement of 64bit/s and assuming the sending of one packet per second, the TCP overhead reaches almost 10000%. We developed an example bit counter for DARPA that only counts payload data and not the overhead of TCP packets. This also allows a fair competition, where teams that have a lot of packet loss do not get unnecessarily penalized.

The network layout is depicted in Figure 9. For the VRC competition, OSRF and DARPA provided the DRC simulator (Gazebo) to simulate Atlas and the environment. During the competition, the DRC simulator runs on a DARPA controlled cloud computer.
Our control software communicates with the DRC simulator over a ROS layer that allows synchronization between the simulator and controller. The DRC simulator provides sensor data updates at 1000 Hz, while our control software provides joint torque commands at 200 Hz.

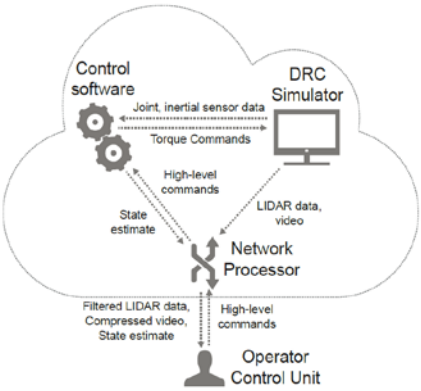

Figure 9. ***Network layout for the IHMC VRC entry. The control software communicates with the DRC simulator through a high bandwidth connection at a rate of 1000 Hz. The Operator Control Unit communicates through a network processor at low bandwidth and high latency.***

Processing of remote sensor data, i.e. LIDAR data and camera images, is done on the network processor. Separating the low-level control from remote sensor data processing avoids potential delays due to garbage collection and usage of all resources. The network processor is also responsible for the filtering, compression, and decompression of the data that is sent to the Operator Control Unit (OCU). The network processor sends the OCU the left camera image and the camera pose with respect to the world, the raw LIDAR scan ranges and the current LIDAR sensor pose with respect to the world, as well as joint angles and the pose of the pelvis with respect to the world. This allows us to recreate the 3D environment, to be displayed to the user.

TCP guarantees that all packets arrive at the destination, so we can assume that we have a 100% reliable link, simplifying the design of the communication protocol. Every message consists of a 1 byte header and a variable length payload. Individual commands send to the controller can be split into control commands and pose commands (position and orientation). Control commands are generally send as only the header, while pose commands are sent relative to the pelvis in low resolution (6 - 14 bits per element). This allows us to compress a pose to about 64 bits, resulting in the ability to send approximately one pose message per second. The pelvis pose is buffered on both the network processor side and the OCU side, each having a one byte identification. Thus, the pose commands are always exact, no matter the communication delays and movement of the pelvis.

From the network processor, robot state, video data and LIDAR data are send to the OCU. Robot state consists of the pelvis pose and the joint angles in a low resolution (10 bits) format. Video data is compressed using the H.264 codec, and the frame rate and compression ratio is user selectable. At one frame per second and a 240 by 240-pixel resolution at the lowest quality setting, video data only takes 10 kbits/second. Both the network processor and the OCU hold on to a tree structure describing the 3D world as constructed from the LIDAR data. The network processor tests if each individual LIDAR ray changes the tree structure, and only sends over the ray index and length if it does change the tree structure. The LIDAR pose is send in high resolution to the OCU, allowing

a good reconstruction. This filtering reduces the LIDAR data rate towards 10 - 20 kbit/second depending on the terrain. The simulation uses lower resolution textures than real world and we expect that the data rates will increase when using physical sensors.

3.1.14 Autonomous Driving

We investigated various autonomous driving algorithms during the competition. Using the cheat mode in Gazebo, we were able to test these algorithms before the car ingress/egress behavior was complete. All autonomous driving algorithms are trajectory based. Trajectories are defined as a desired steering wheel angle and a time-based sequence of desired throttle and brake pedal positions.

To learn the outcome of a trajectory, we executed the trajectory in the simulation and examined the vehicle's true motion. This way the vehicle could be quickly retrained and repeatability of different motions learned. The results of learning were saved to a file for later use.

The driving algorithm uses the camera image of the left eye to generate an overhead map of the road. The overhead map was created using stereo image processing. First the dense stereo disparity was found. This stereo disparity is used to associate points between the left and right images. The best fit homography was found using RANSAC. From the homography a rigid body transformation to the ground plane was found. Once this transformation is known a ray can be cast from the camera to the ground to create a synthetic overhead view.

Two autonomous algorithms were created. Both worked by segmenting the overhead image in road and non-road components. Segmentation was performed using Gaussian color models created from manually labeled images. After segmentation was done a c-space cost map is created by expanding obstacles. Cost of a trajectory is computed by overlaying the trajectory on cost map and integrating.

The first autonomous algorithm would select a trajectory and execute it fully before selecting the next one. This was a fall back just in case the robot would move around too much while driving. Another autonomous algorithm continuous selected new trajectories. Validation of both approaches was done on the DARPA provided map in addition to several more challenging map which we created.

Unfortunately, time constraints and artifacts in the physics of the simulation made in not possible to use the autonomous driving algorithms during the competition, and we fell back to pure tele-operation using the arrow keys on the keyboard instead.

3.1.15 Emergency Recovery Behaviors

We developed a small library of joint trajectories for use in emergency cases during the competition, namely losing our balance and falling down. The library included scripts for rolling over, repositioning to look around, crawling, scooting away from obstacles,

getting out of the car, and standing up. Multiple ways of achieving each behavior were implemented.

To select the desired joints, we interfaced a slider board with our simulator with an editor that allowed us to directly control either joint angles of the Cartesian locations of hands, feet, pelvis and chest. Each pose represents the desired joint angles to be reached at one second intervals, with a variable pause in between. The editor showed the center of mass and support polygon visuals since we needed the robot to be statically stable at all times. The desired joint angles were fed to a direct joint based PD controller where position gains were set much higher than gains used in the whole body controller.

3.1.16 Operator Interface

One of the challenges faced in the VRC was efficient use of data. Since our approach relies heavily on human involvement, it was necessary to provide sufficient data to the human. There are two main sensors for world state; the stereo cameras and the LIDAR. We did not use the hand cameras. For the stereo cameras we simply provided a means for the operator to manually set the resolution and frame rate based on the task. For example, the walking task typically could use a low frame rate and the driving one used a higher frame rate.

The LIDAR data was more complex. We needed long range information for planning, and we needed high fidelity information for manipulation, but we wanted to avoid sending massive amounts of data constantly. Our solution was a data structure that combined a quadtree and an octree. Our LIDAR data was put into a quadtree on the Field computer. The quadtree was used to provide a heightmap of the terrain for walking, as shown in Figure 10. Anything above the terrain was pushed to an octree. The octree represented all obstacles and objects in the world. An example is the gate posts visible in Figure 10. This combined data structure was then used to filter what was sent to the operator. Only changes to the data structure get sent, so static aspects of the world do not generate any additional messages.

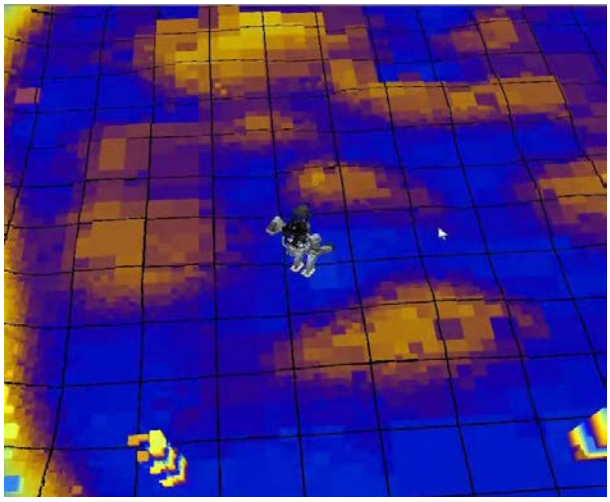

Figure 10. ***Height map of terrain based on LIDAR.***

For manipulation, a very fine-grained octree was necessary, so our octree resolution varied based on distance from the robot. This is depicted in Figure 11, where the hose is shown in higher resolution than the walls and legs of the table.

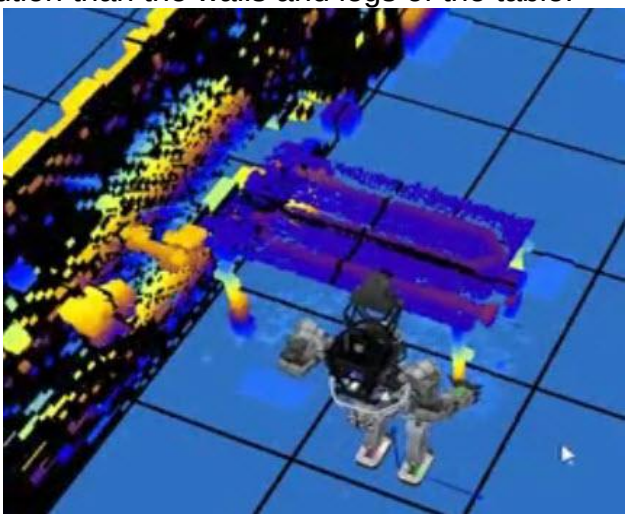

Figure 11. ***Variable Octree resolution for the hose task.***

3.1.17 Walking Task

We rely on the human operator for all planning and the robot controller for all dynamic balance. A screenshot of the user interface during footstep planning is shown in Figure 12. The operator provides footstep path specification using a control ring, which allows specification of the goal location, walking direction and step size. Additionally, the operator can manipulate individual footsteps. It is up to the operator to avoid difficult terrain. To support this, we provide a footstep riskiness indicator, in which footsteps are colored based on their pitch and roll, and based on their distance from the previous footstep. This allows the user to quickly judge the likeliness of falling for a given target location. If there are any risky footsteps, then the user can either choose a completely different path to take or slightly move the final desired target around until the risky footsteps are adjusted to fall on less risky steps. We find that this simple way of giving the operator feedback and control works very well on the walking task.

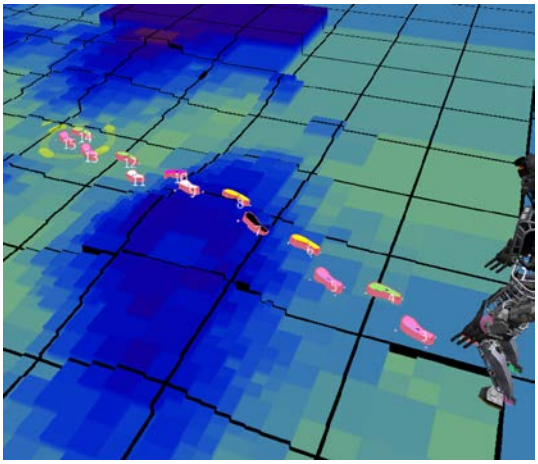

Figure 12. ***Footstep planning in the user interface. Footsteps snap to the LIDAR world map, and are always planned along straight-line paths. The control ring at the end of a footstep path can be dragged to change the target, and can also be used to change the direction in which the robot is facing during locomotion, allowing sidesteps and backwards walking. Footstep colors signify various levels of perceived danger.***

3.1.18 Hose Task

We use scripting to assist in much of the manipulation task. However, our scripting is dynamic and flexible. We provide visual manipulables that allow the operator to specify the location of the object to be manipulated, and all scripts are referenced to the object, instead of being relative to the world or robot frame. This enables the operator to adjust manipulation at runtime and account to errors in state estimation or task execution. Figure 13 shows an example of the hose manipulable used for picking up the hose. We have similar manipulables for the spigot and valve. In addition to grasping, we use these manipulables for alignment. They allow the operator to align the nozzle to the hand or the hand to the spigot with a single click. This makes the aligning task much simpler for the operator.

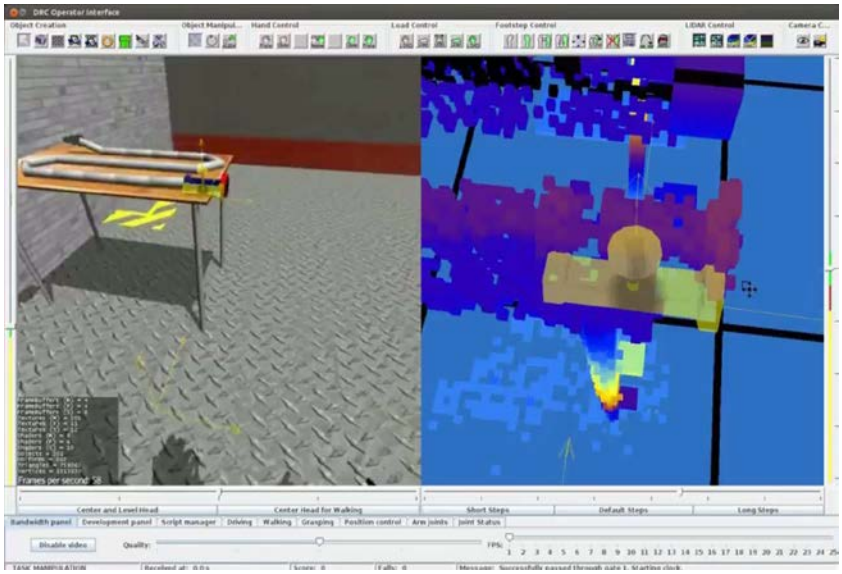

Figure 13. ***Hose manipulable.***

We also provide visual Inverse Kinematics (IK) solution graphics. The Atlas arm configuration has a limited dexterous workspace and is very difficult for an operator to comprehend. We provide visual indications to the operator when the specified hand position is valid (green in Fig. 14) and when it is invalid (red in Fig. 14).

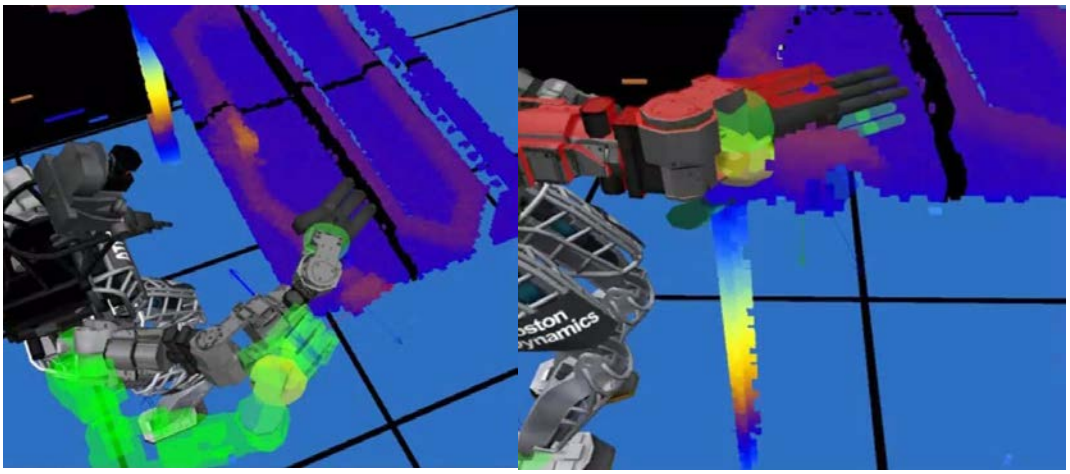

Figure 14. ***Graphic depiction of valid (left side) green IK solutions and invalid (right side) red IK solutions.***

3.1.19 Driving Task

The hardest challenge was getting into the car. We use pose scripting, described elsewhere, to enable this. The ingress was accomplished using pose manipulables to generate scripts and to validate and adjust them at runtime.

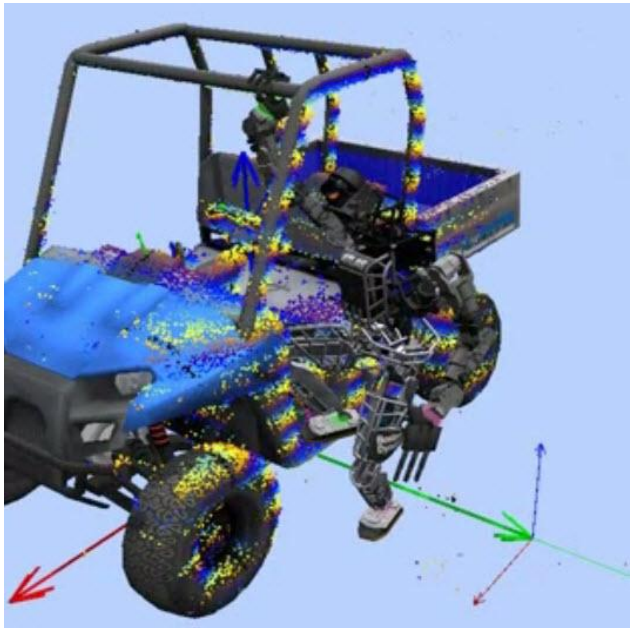

Figure 15. ***Vehicle ingress.***

Once in the car, we provide a very basic driving interface (Fig. 16) that allows the operator to drive the car much like a video game. The operator can control steering, gas and brake. We had some autonomous approaches, discussed elsewhere, but ran out of time to fully integrate and test them.

Figure 16. ***Driving interface***

3.1.20 Scripting

Perhaps the highest abstraction level feature we created in the UI was to support recording and executing a sequence of robot commands as a modifiable script. Once we press the record button, the network processor serializes all communication packets received from the UI into an Extensible Markup Language (XML) script file. Any pose commands are transformed to a user-specified reference frame before serialization. This feature allows e.g. playback of a vehicle ingress script relative to the user positioned vehicle graphic and thus allows identical ingress behavior across multiple runs with differing vehicle poses. Figure 17 shows the user interface while performing vehicle ingress using a script.

Figure 17. ***Snapshot of the user interface where the operator is preparing to command the robot to exit the next desired pelvis pose in a script, as visualized by the semitransparent, yellow 3D model of the pelvis.***

To keep our approach more general and to reduce automation surprises, instead of purely playing back scripted commands, each command in a script shows up in the user interface as if the operator just created the command. This allows the operator to see what the robot is going to do next and allows online adjustments to the command if necessary to make it appropriate for the current state of the robot and world. The operator can also choose to skip individual commands, or to step backward through a script in order to repeat previous commands. This greatly improves robustness relative to pure script playback. These user interface features together provide a stable, efficient, and trustworthy UI for controlling a humanoid robot and make effective use of the human-robot team.

Additionally, to tune the scripts even more, we developed a script editor interface (see Figure 18). This interface provides the possibilities to alter time steps, delete pauses, show and alter the loaded file in its XML format, extend loaded scripts with different scripts and save the newly created scripts.

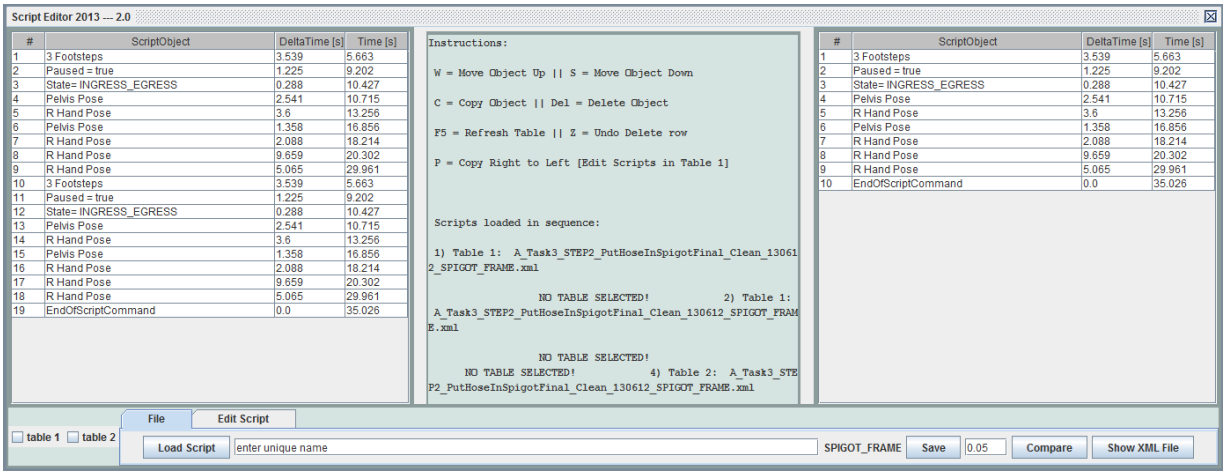

Figure 18. ***Script Editor Interface with three total scripts loaded in.***

3.1.21 Operator Training

The Virtual Robotics Challenge was not just a programming challenge. In some regards it was also a sporting event, in which real time human performance mattered as much as the nine months of software development leading up to the challenge. Therefore, we put a lot of effort into operator training and practice. We allowed anyone on the team to try out to be an operator. To do so, a team member needed to run their task a large number of times and score 4/4 points on it with high probability. Any time we changed the software, we required the operators to perform the test challenge again and score 4 points. In addition to the example challenges provided by DARPA, we also developed additional challenge environments. For the driving task, we made extra driving courses, for the walking task we made mud pits of various viscosity and more challenging hills, and for the hose task we rearranged the location of all the components, including putting the spigot on the right side of the table.

Since we had a large number of qualified operators, we made it so that no operator operates more than one challenge event so that they could focus on their one event. For each event, we had the group of qualified operators decide who would go first for each event and who would be the “co-pilot”.

During the actual VRC competition, we operated conservatively, but we tried to have each operator “play like they practice”. Once the initial jitters of each event passed, we found that we were able to successfully operate the tasks, in part based on our intensive operator training.

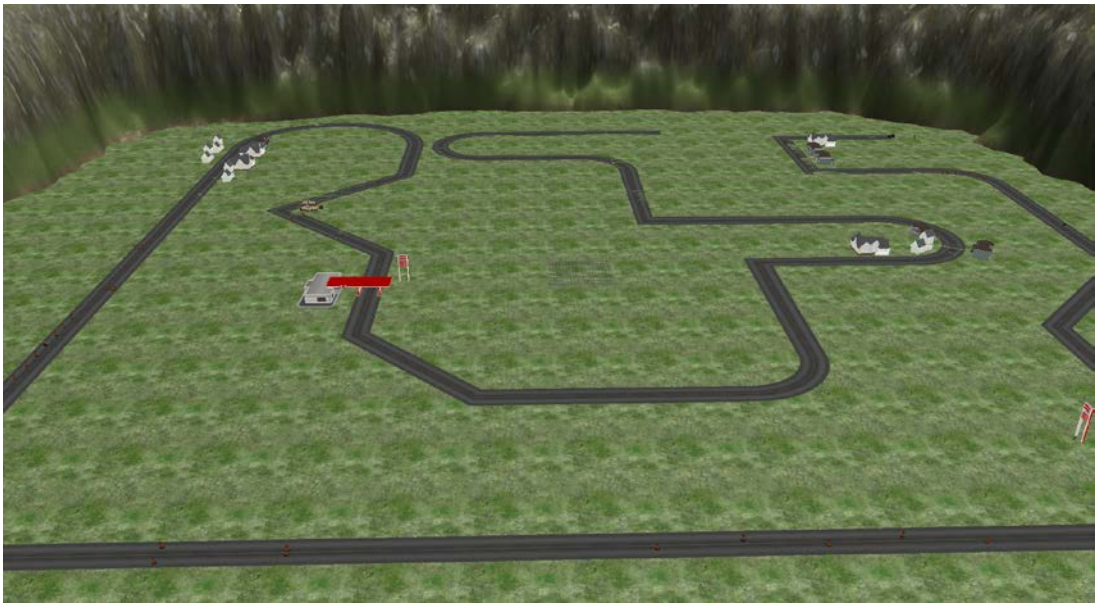

Figure 19. ***Custom driving course made to test our driving controller. Notable obstacles include houses obstructing the view of barrels, barrels obstructing the view of the sides of the road, and a “drive-thru” gas station.***

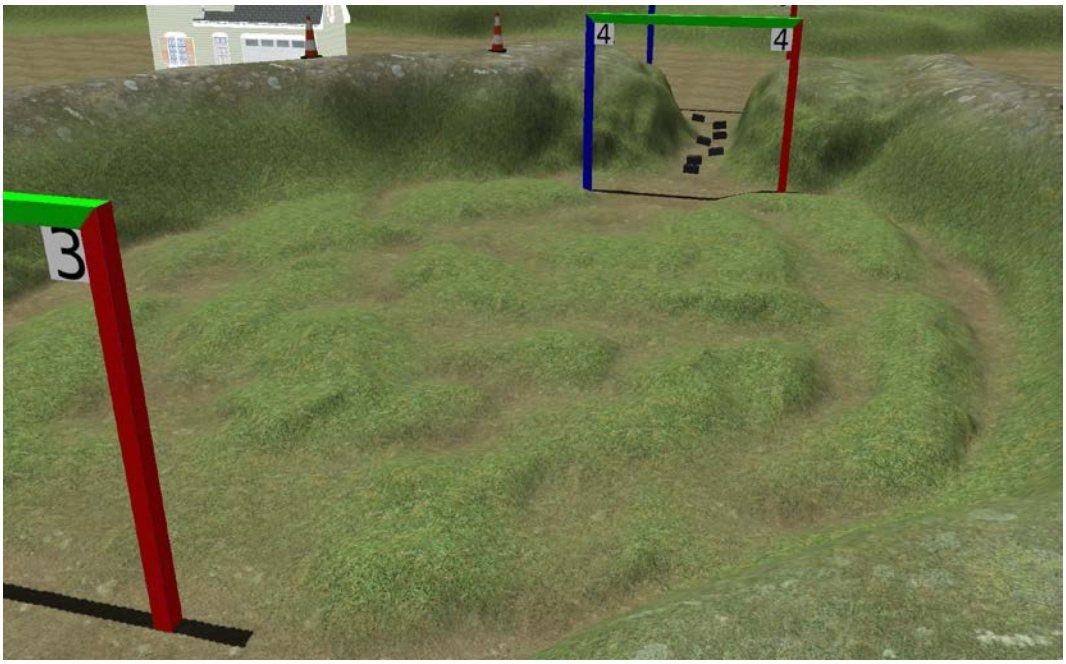

Figure 20. ***Custom walking course with steeper hills and a narrow, cinder block-laden canyon.***

Figure 21. ***Custom manipulation task with a larger and more massive hose connector, lower table, valve and standpipe placement reversed, and smaller valve with higher damping parameter.***

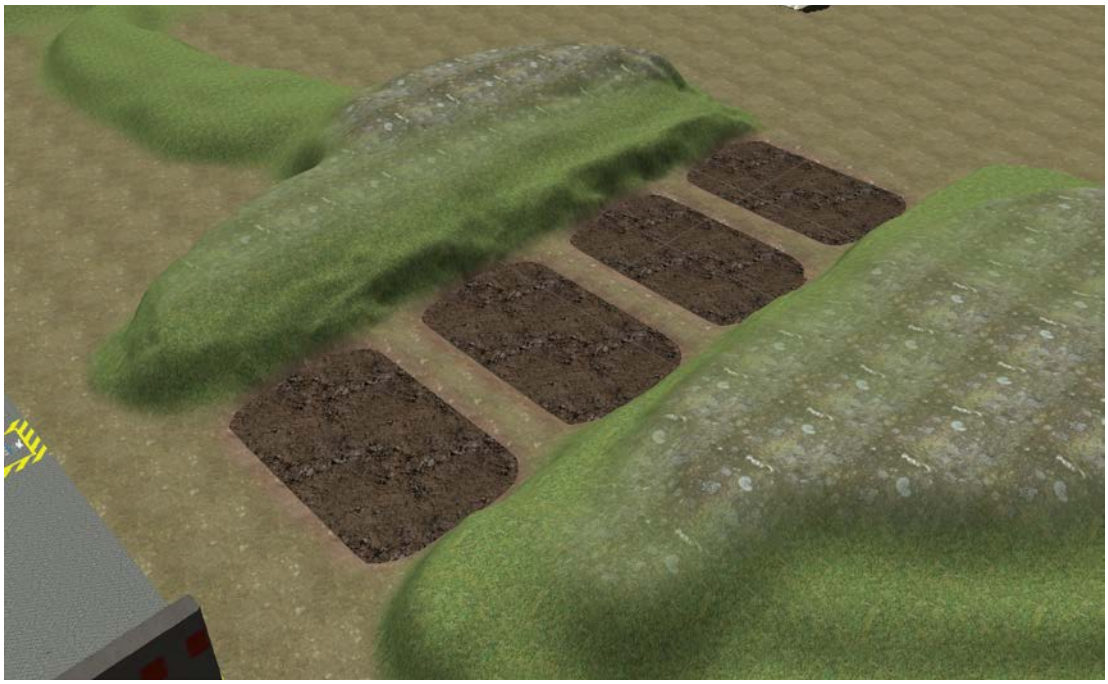

Figure 22. ***Custom walking task consisting of four mud pits with increasing friction and damping parameters.***

Figure 23. ***Custom world with heightmap made from the face of our Principle Investigator/fearless leader.***

3.1.22 Virtual Robotics Challenge Results

We received a total of 52 out of 60 possible points, making us the top finisher in the overall competition. We were most successful on task 2 (walking) while task 1 (driving) was the most challenging. The following table shows our scores on individual tasks as well as qualitative evaluations.

Run #	Task	Score	Notes
1	1	**1**	Arm stuck in the back of the car, recovered, ran out of time
2	3	**4**	Fantastic run, spigot close to table, high friction valve
3	2	**4**	Unexplained fall on hills, got back up and finished
4	2	**4**	Completed cautiously with no problems
5	1	**1**	Problems with Polaris
6	2	**4**	Took it cautiously with no problems
7	3	**4**	GUI crashed but recovered and finished with no more problems
8	3	**4**	Hose unscrewed from spigot
9	3	**3**	Fell over, ended up under the table, got back up and finished.
10	3	**4**	No problems
11	2	**4**	No problems
12	1	**3**	Fantastic Recovery
13	1	**4**	No problems
14	1	**4**	No Problems
15	2	**4**	Final run, ran very well

Task 1

The driving task gave us the most trouble of the three challenges. Car ingress was very systematic thanks to our ability to record and playback scripts in the car frame. Getting ready to switch from the car ingress controller to the driving controller and making the switch itself gave us the most issues. Problems included high-level controller issues and running out of time in preparing the robot configuration and Polaris for driving the course (e.g. not having a firm grasp on the roll cage and steering wheel, being in reverse, not disengaging the handbrake). Our first two runs had these kinds of problems, giving us 1 point for each run. In the last three runs, we were able to successfully drive the whole course, and got perfect scores on the last two runs. Although we successfully completed

the last two runs, our last run was truly perfect in that our car egress was completely controlled, with no falls.

Task 2

During the challenge, we were most confident about the walking task. The walking algorithm, the footstep riskiness indicator and the many hours of practicing ensured that we could complete this task with a high probability. Despite falling in one of the runs, we scored the maximum amount of points in all of our task two runs. The robot was able to get up in the hill section of the task, and complete the task, getting all the points for the team. This recovery reflects how robust our robot was in different environments to which it was exposed.

Task 3

The ability to record and playback scripts as well as hours of practice was pivotal to our success in Task 3. We created a variety of scripts for picking up the hose, aligning it to the spigot, connecting the hose to the spigot and finally turning the valve. Overall, we received a total of 19 points on the task. Although the scripts did make the task much easier, we did run into a few problems, such as unexpected falls. In one case, we were unable to recover because Atlas was stuck under the table and ran out of time. Luckily, the other time we fell, we were able to get back up thanks to the special teams' quick thinking and "getting up" scripts.

3.2 Phase 2: DARPA Robotics Challenge – Trials

The DRC Trials consisted of eight tasks designed to mimic actions performed by first responders in disaster environments. For each task a robot is allotted thirty minutes to complete the objective. The operator was required to be out of line of sight of the robot at all times. Additionally, DARPA provided bandwidth constraints by introducing a network shaper which oscillated throughput between “good” (1Mb/s and 100ms delay) and “bad” (100kb/s and 1000ms delay) communications. Each task has individual criterion for awarding a total of four points, with incremental scoring for the first three points and a bonus point for task completion without intervention. A brief overview of the eight tasks is as follows:

1) Vehicle: The robot begins inside a Polaris Ranger XP 900 and must drive through a cascading obstacle course then exit the vehicle and travel to a designated exit.
2) Terrain: The robot must travel over a pair of slanted ramps and over cinder blocks in a zigzag pattern, across flat-top steps, and across angle-topped steps.
3) Ladder: The robot must ascend either a 60 or 75-degree ladder approximately 2.4 meters high.
4) Debris: The robot must remove 10 pieces of debris, walk around a metal truss, and travel through a blocked doorway.
5) Door: The robot must open and travel through three doors: a push style door, a pull style door, and a pull door with a weighted self-closing mechanism.
6) Wall: The robot must use a cordless drill or rotozip tool to cut a prescribed triangular shape out of a designated area of sheetrock.
7) Valve: A robot is required to close three valves; a 3” CS Ball Valve, a 9” diameter Gate Valve, and an 18” diameter gate valve.
8) Hose: A robot is required to grab a fire hose and attach the connector to a spigot located along an adjoining wall.

All teams using the Atlas robot are required to run their code off-board on what is called the “field” computer (Fig. 24). This computer represents what would eventually be running onboard. It contains all control algorithms and state estimation. It also is the network processor for all data going to the operator. The operator uses an operations computer that is remote and out of the line of sight of the robot at all times. DARPA provided a bandwidth limiter to constraint communications in way they deemed would mirror what would be realistically available in an operational scenario.

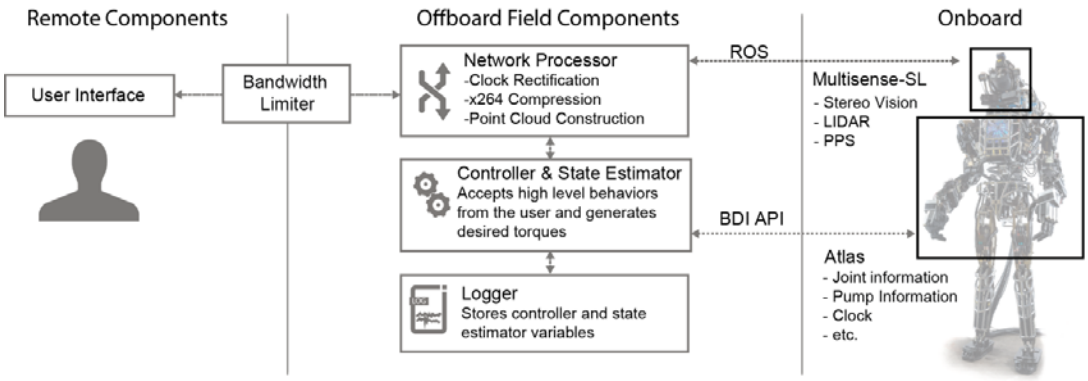

Figure 24. ***System Architecture.***

The primary challenge in transitioning from the VRC to the DRC Trials was in understanding and addressing the differences between simulation and hardware. Several issues discussed are common issues in robotics and others are specific to the Atlas or the competition. We will discuss how each issue impacted performance and what we did to mitigate the effects.

3.2.1 Sensing

For the DRC, the operator was required to be remote and out of the line of sight of the robot at all times. This meant that the operator's awareness was completely dependent on accurate and timely sensor data. This was not a problem for simulated environment of the VRC, but posed several challenges on the real Atlas hardware for the DRC Trials. These issues mainly dealt with synchronization and error.

3.2.2 Sensor Synchronization

During the VRC, the LIDAR and camera outputs correlated with the Virtual Atlas's clock, providing perfect synchronization. Moving to hardware this was not the case. The camera and LIDAR data are provided by the MultiSense-SL head independent from the other sensor data provided by Atlas. The head used a Pulse Per Second (PPS) signal to timestamp the sensor outputs. The Atlas robot stores a timestamp that correlates to the last time it received a PPS pulse from the head. By modifying the PPS output to broadcast its internal time instead of the increasing value of the PPS, we were able to calculate an offset between the robot time and the MultiSense-SL time that updates every second when a new PPS pulse is generated. This offset was used to correlate LIDAR packets with the robot's sensor data and in turn allowed the accurate placement of LIDAR points with respect to the robot.

3.2.3 Accounting for LIDAR Rotation

The Multisense-SL's Hokuyo rangefinder is continuously rotated around its axis by a full rotation spindle. Thus, the transform from the LIDAR to robot reference frame is continuously changing during a single sweep. During the VRC these scans were an instantaneous snapshot and a single sweep had a constant transform. To compensate for the continuously rotating LIDAR, the transform of each point is generated by interpolating between the pose at the beginning and end of each scan. This interpolation

of the base of the Hokuyo is done in world frame, incorporating movement of the robot base in the interpolation.

3.2.4 LIDAR Shadows

While LIDAR data was very clean in simulation, the actual Hokuyo LIDAR generated noise along the edges of objects. LIDAR measures range using a beam of laser light. If the beam hits multiple objects, then it returns a range which is a combination of the range of both objects. This structured range appears as a shadow, as shown in Figure 25, and makes the edges of objects difficult for the operator to see. This is especially true in cluttered environments; such as the debris removal task. This makes task that involve accurately placing an end effector behind something like the valve shown in Figure 25 challenging. Our rainbow depth color scheme, shown in the right side of Figure 25, provided the operator with some ability to estimate thickness.

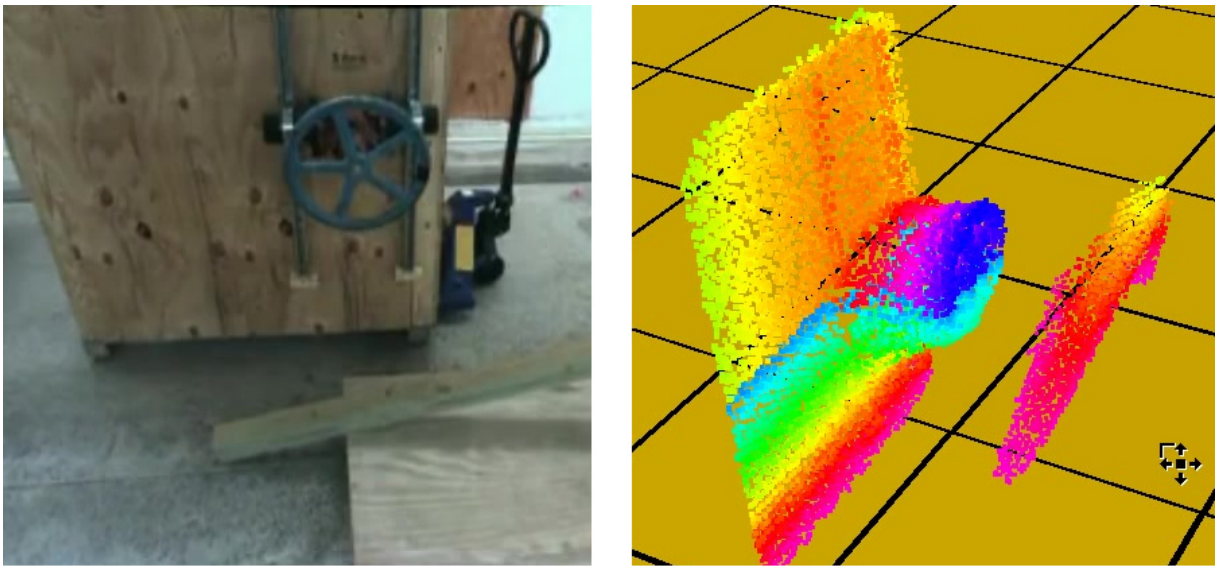

Figure 25. **Left: Camera image of valve. Right: Point cloud showing LIDAR shadow. Note how edges of the valve have false points going back to the wall. The rainbow color depth map provides a mechanism for the operator to estimate the actual thickness of an object.**

3.2.5 Joint angle offsets and calibration

While joint angle offsets were not an issue in simulation, they are a critical issue on hardware. Joint angle measurements of the robot's arm from the onboard sensors were only crudely calibrated by the manufacturer. As a result, there was an offset between the actual joint angles and the measured joint angles. In addition, the offset was conditional upon the actual orientation of each joint in the robot. The net effect of the offset was an inaccurate robot model in the user interface which would significantly degrade the operator's ability to control the robot with any precision.

3.2.6 Arm Calibration

We also worked on arm calibration. The arm joint angles were not accurately calibrated by BDI, thus the desired and actual wrist position had an offset. We implemented several methods of determining the angle offset of the arm joints, based on vision algorithms. A calibration target was fixed onto the end of the left and right arm. An operator would then move the arms into different positions in which the target was visible in the left stereo camera. The pose of the calibration target was then estimated using its known geometric configuration. Accuracy of the pose would depend on the distance and orientation. After several target poses were collected a batch optimization was performed to compute joint angle biases. In addition to the joint angle bias the transform from the last joint to the calibration target also had to be estimated. After optimization, the residual error was reduced significantly. Additional work is needed to further improve and automate the process. An apparent lag between the measured joint angles and observed location of the calibration target was also observed. This lag should be compensated for during calibration and potentially estimated so that it can be accounted for doing control operations.

Images were collected continuously while moving the robot's arm (Fig. 26). Images which were badly corrupted by motion blur were mostly removed automatically due to the target not being detected, with the remaining few removed after manual inspection. The total sum of square pixel error was reduced by 2700 times after optimization. Errors were strongly correlated with the arm being stationary versus moving and even better results are expected if this is compensated for.

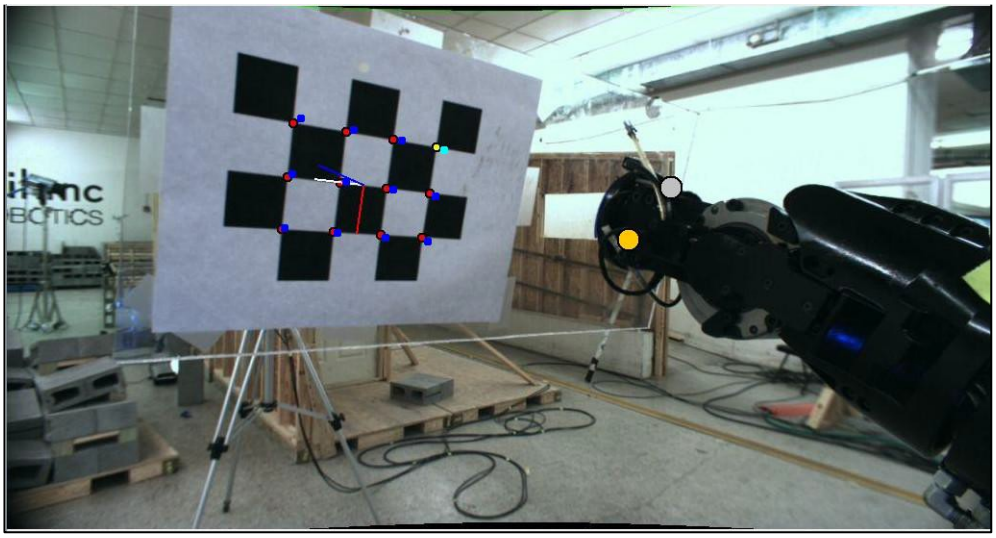

Figure 26. ***Arm calibration routine showing the residual error after bias estimation. Red dots are the calibration points on the target. Blue dots are the predicted location given our kinematics model and the bias estimate from batch optimization. The gray dot is the end effector position before any correction and the yellow dot is the change after correction.***

3.2.7 Walking

Models provided by OSRF changed frequently with little verification or explanation. Our force control requires accurate models to function well. Errors in mass, joint offsets, and link meshes indicated a need for verification and corrections, but time constraints and lack of information forced hacks that were gradually removed later.

Force control of the arms joints was extremely limited and combined with large amounts of stiction in the arm joints led to our decision to use low-level PD position control on the arms. While the joint PD controllers tracked desired trajectories very accurately, trajectories were followed in an extremely jerky fashion. We switched to a hybrid controller that combined force control based on the desired arm joint accelerations with a PD controller based on desired arm joint positions and velocities. While having a slightly larger error than low level PD control alone, the amount of jerk during trajectory execution is minimized. During the tasks, this reduces unintentional collisions with the environment.

Previously, we would manually transition from stand prep to stand mode by scaling the corresponding control gains. During the trials, the operator has limited information from the robot therefore in order to increase reliability and repeatability this process was automated.

The real robot has backlash, communication latency, and link flexibility causing the controller to misinterpret what's the robot is actually doing with a certain delay. That can cause instabilities often resulting in rumbling. We were able to partially cancel rumbling effects by filtering sensor measurements, adding compensation in sensor measurement for backlash, and reducing the damping gain. Due to the use of Jacobian matrices to compute desired joint accelerations, the controller is sensitive to singularities such as straight leg configuration. We have implemented two components to overcome this issue. The first is a singularity check that, if necessary, induces a singularity escape using the Jacobian's null space. The second is an added controller to limit the acceleration of the knee when it gets close to the straight leg configuration.

Since the knee actuators are weak, they represent a critical failure point on the robot. We experienced several instances in where the knee would collapse. These instances typically happened if the knees were bent too much while under load. This is particularly true on the debris task because the robot will be required to regularly operate quite low in order to pick up low-laying debris pieces. To prevent the robot from falling we implemented a controller that limits the support knee range of motion depending on the load sensed. There were other less frequent cases in where the knee would collapse while operating at normal robot height. These failures were harder to diagnose and prevent because not joint limits were being reached. This type of failure was experienced during the Valve task at the DRC trials.

Reliability of foot roll off while walking was also improved by adding the capability for the walking controller to step down from objects without knowing where ground is. This was achieved by improving the ICP planner and the robot height controller resulting in

a more robust touchdown. Another important improvement was introduced with the implementation of a PD controller to hold the foot position when a foot is barely loaded resulting in reduced foot slippage. On top of this, online tuning of controller gains and parameters affecting touchdown was performed in order to achieve softer touchdowns.

3.2.8 Hands

Development and testing of manipulation control was done independently (e.g., bench testing) and then integrated with existing manipulation functions already present in the UI. The hands were a separate software system that occasionally required a hard reset in an event of failure.

The iRobot hands proved viable but troublesome for several of the DRC tasks. Hardware malfunctions and maintenance schedules forced downtime and limited the practicable tasks whilst the hands were being serviced. Additional hands would have allowed for a more flexible workflow. Due to limited quantity, iRobot did not have additional hands that they could supply, either on a purchasable or loanable basis. These maintenance constraints and the unreliability of the iRobot hands forced us to adopt a hand hook hybrid model. Depending on the manipulation requirements of a task, Hook hands were used to limit failure points. The debris task was the most manipulation intensive and required two iRobot hands, while an iRobot hand- hook combination was used for the wall, hose, and valve. Two hooks were used during the door task and no hands were used during terrain and ladder. The image below depicts the iRobot hand-hook combination during the wall task.

Figure 27. ***Hook hand and iRobot hand used in combination.***

The iRobot hands were prone to tendon failure caused by tendon abrasion and internal knot failure. To limit the maintenance requirements of the iRobot hands the frequency of tendon breakage had to be reduced. The tendon abrasion was caused by a chamfer inside a routing hole just below the finger mounting plate. During maintenance, a

routing tube can become misaligned causing the tendons to abrade against this chamfer. Rounding down the chamfer proved to greatly increase the durability of the tendon and extend tendon life. The image below represents a modified routing hole. The chamfer is located inside the feeder holes positioned at three and nine o'clock from center. The limited workspace and visual access required great care during modifications.

Figure 28. ***Modified routing hole of iRobot hand for reduced tendon breakage.***

For many of the tasks Atlas's workspace is unforgiving. The 6-degree of freedom arms hinder fluid movement in a human environment while the joint limits and joint failures in the torso and legs retard the range of motion during crouch maneuvers. These restrictions prevent Atlas from performing ground level manipulation and reduce efficiency when faced with tasks requiring a wide area workspace. A modifiable forearm extension was developed with 4", 6" and 8" extension options. The forearm extension allows for ground level manipulation and extended reach. The extensions were tested for all of the manipulation tasks, but only used during the debris task during the DRC trials. The 8" extensions provided the most benefit and were used during the task.

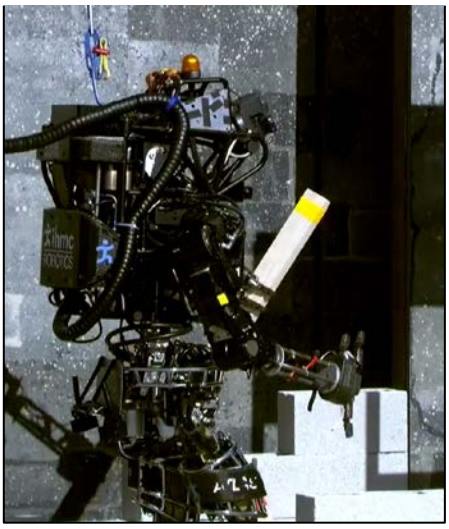

Figure 29. ***Atlas with 8-inch arm extensions during the debris task.***

The weight displacement from the extensions also provided a means to escape the back failure (a non-responsive pelvis pitch with the torso hunched forward). In an

instance of back failure, rotating the arms behind the body moved the upper torso CoM backwards, allowing the servo to push the torso erect.

3.2.9 User Interface

The UI required significant improvements and modifications to transition from the VRC simulation to Atlas hardware for the DRC.

All Atlas sensors need to be exposed through the interface, as well as the controls for managing the various sensors. The operator is able to control LIDAR filtering parameters from the UI. We improved the view from fisheye cameras in our virtual reality interface and parameterized quality and frame rate for operator control. Additionally, camera properties are adjustable such as colorization and exposure time to operate optimally under adverse lighting conditions. Fisheye camera drivers were developed as a ROS node with camera properties modifiable using ROS's tools. Bandwidth was reviewed as the bandwidth used by the fisheyes is very expensive and required compression high enough to limit the effectiveness.

One of the cameras was plagued by hardware failures that were later found to be bad connectors and were eventually repaired by BDI. We also improved our Quadtree representation of the LIDAR data by reducing noise.

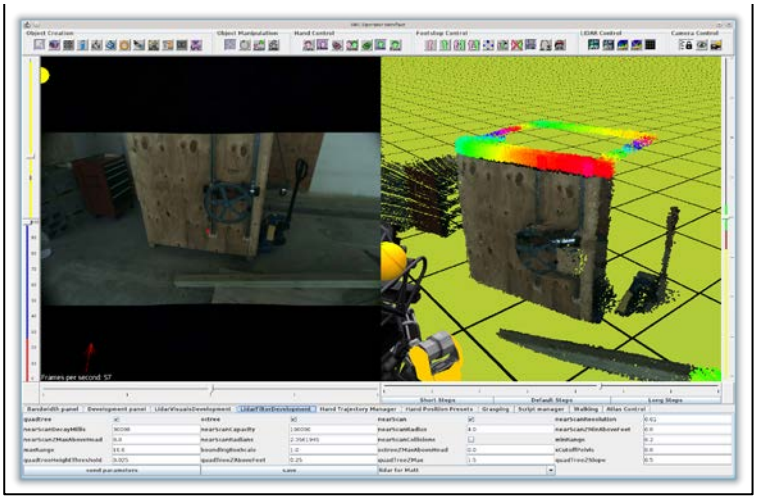

Figure 30. ***Our UI with the live camera image (left side of UI) wrapped by the fisheye image (too dark to tell in this image). The lidar can be displayed as a quadtree, point cloud or a colorized point cloud (shown on the ride side of UI).***

The walking control interface components from the VRC were sufficient for the Trials and remained unchanged. However, the arm control components were insufficient. This was mostly due to the sensing noise/errors encountered on Atlas that were not accurately modeled in Gazebo. To account for this, it was necessary to present more information, so the operator could make an informed decision. For example, we enable LIDAR collisions with the body to see joint offsets between the reported joint position and the LIDAR sensed joint positions.

Walking over slanted cinder blocks required some interface support. From the UI perspective, small modifications on footstep placement and manipulation were added in order to improve this robot capability. Some of these modifications included footstep to surface snapping and individual footstep attitude and position manipulation.

We added some additional user interface features for the new tasks of the DRC Trials. The operator has a better way to align body when going through doors by using Augmented Reality (AR) techniques and the different available cameras. This works by first loading an AR model of the door into the main viewport. The model is then manually aligned by comparing it to the real camera image in the background. Any AR model placed in the main viewport is internally represented by a node whose parent is the world's main node (also known as root node). All nodes are automatically updated using the robot's filtered IMU data. This allows the operator to simply drop, align and temporarily forget AR models that represent interactive objects in the main viewport for a prolonged amount of time. As the robot moves around the IMU data is used to translate and orient the nodes in the viewport accordingly. Due to the nature of the IMU device, drift accumulates over time. Therefore, the AR model will visually drift away from the background image (offset error build-up) requiring the operator to eventually manually re-align the AR model again. Typically, most AR models are known

as interactive objects. As the name suggests, an interactive object is some object with which the robot can interact in some way or another. Interactive objects typically contain a varying number of child nodes that represent interactive points that are used to help the robot navigate through the interaction process. The benefit of interactive objects is that the operator can simply click on them to generate some implied robot-interaction. For instance, in the case of the door, after aligning the door interactive object (AR model) the operator can simply right click on the object to command to robot to go through the door. This command then utilizes the object's interaction points to make the robot navigate through the door. Since interaction points are relative to their parent, the main interactive object's node, the operator can easily move or rotate the interactive object around knowing that the parent-child relationship between the object and its interaction points is still maintained.

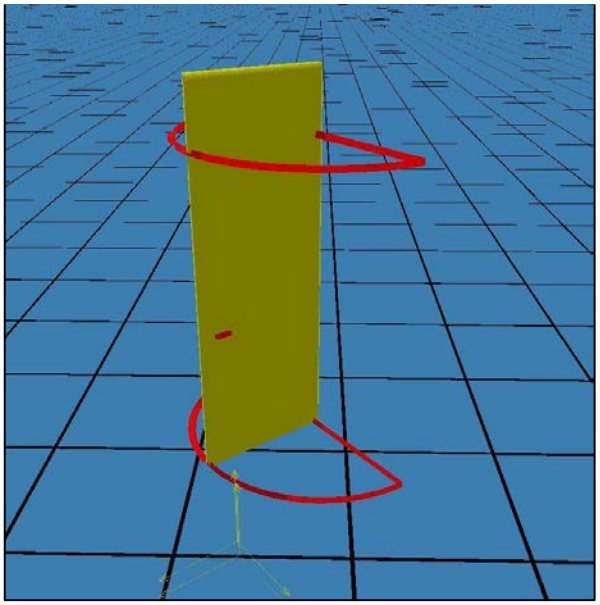

Figure 31. ***Door interactive object in our UI.***

In order to simplify and speed up the operator's duties on the field an interactive user-interface menu system was developed. This menu provides quick-access to task-relevant interactive objects such as ladders, valves, doors and cinder blocks by interactively overlaying the relevant options on the operator's screen. The user can also kinematically link multiple interactive objects by defining a child-parent relationship between them. This allows a much simpler group object manipulation through the UI by only managing the parent object. The child object can also be manipulated with respect to the parent's origin (i.e. moving virtual hand with a rotating valve). This linking capability can be turned on/off whenever the situation requires it.

As functionality became more reliable, we employed scripting to sequence controls together for more complex behavior. The operator could record and execute scripts to control the robot. This can be done a prior or on the fly. All scripts provide observability, predictability and directability as with our previous work.

3.2.10 Coloring and texture

Even with the higher resolution LIDAR data, it was still difficult to distinguish small features, such as door handles and buttons. To address this, we employed two different techniques. The first was a coloring technique that changed the color of the point cloud elements as a function of the horizontal radius of the point cloud elements to the LIDAR sensor head. This provided adequate differentiation for most objects. The second technique was a more complex texturing method that allowed point cloud elements to adopt the color from the real object. This made point cloud interpretation by the operator simpler and aided in identifying small features.

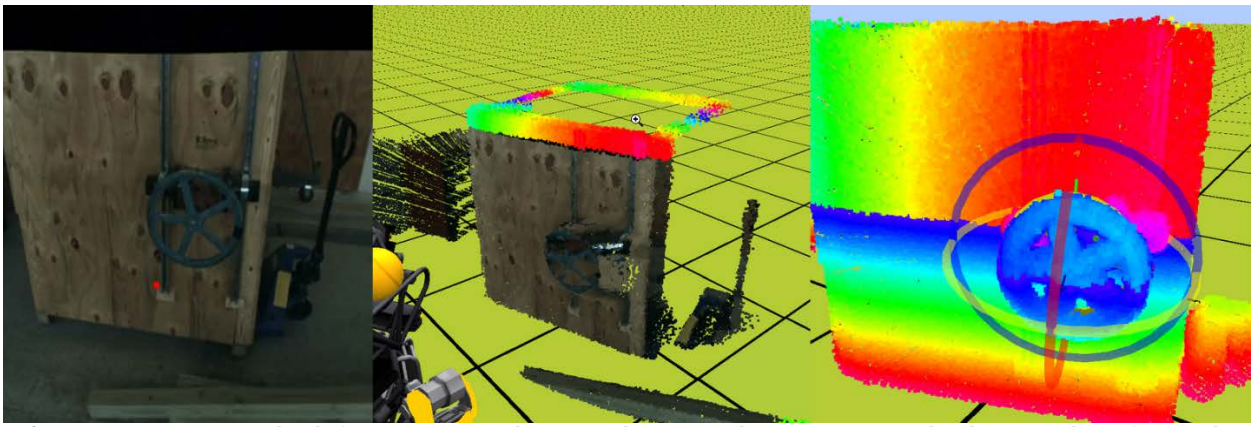

Figure 32. ***Image on the left represents the actual camera image as seen in the user interface. The center image shows the texturing feature on the point cloud data. The right image shows the coloring feature on the point cloud data.***

3.2.11 A wider field of view

During the VRC, we were able to successfully tackle most tasks only using the cameras and LIDAR from the Atlas sensor head. These were displayed to the operator through the First and Third Person-View displays of the UI. During the Trials, it became apparent that certain tasks such as the door task and debris task would require a wider field of view than afforded by the camera. To support this, we integrated the BackFly cameras located on robot's chest. These cameras are equipped with fisheye lens and provide about 185-degree coverage each along the robot's side compensating for the lack of neck yaw in the robot.

This means that when combined, both BlackFly cameras will provide a full panoramic coverage of the robot surroundings. Instead of displaying images from each camera (head and chest cameras) on different windows within the UI we opted for a more complex, but also more rewarding approach that consisted in fusing the video images from all cameras together into the same viewport. This technique consists on projecting the images from each camera onto a virtual dome within a 3D environment. It effectively transforms the typical 2D camera view into a fully interactive 3D view which only uses a single window. This technique provides richer situation awareness information to the

operator in a simplified manner. The operator could “look around” even though the Atlas has no ability to yaw its head. This provided critical awareness of context outside the field of view of the normal camera.

Figure 33. ***Images from head and chest cameras fused together into the first person viewport. The higher resolution rectangle is the normal camera and the rest of the image is from the fisheye camera. This gives the operator the illusion of looking to the right even though the robot cannot yaw its head.***

3.2.12 Understanding joint limits

The tasks of the DRC Trials pushed the Atlas to its limits. To help the operator better understand where the robot was within its operational envelope we provided color indicators. They indicated whether a particular joint is close to or has reached a joint limit. This allowed the operator to better understand constraints and make more informed decisions.

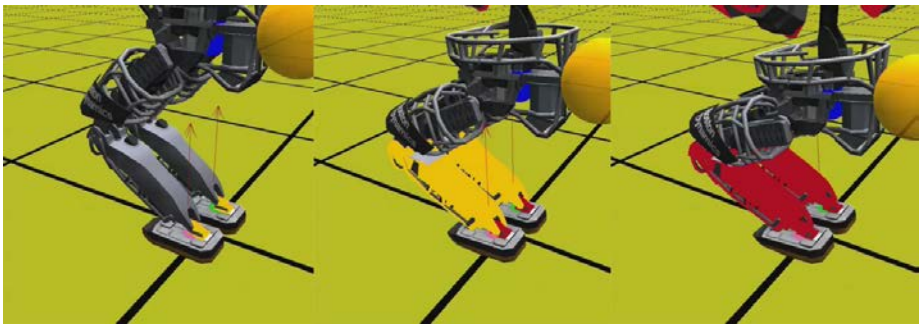

Figure 34. ***Color indicators used to show that joints are reaching their physical limits.***

3.2.13 Infrastructure

We stabilized and improved the communication layer between our software and Atlas to be extremely reliable. Most time was focused on reducing jitter, or large delays in the control code. Main sources of jitter are context switching and allocating and

deallocating memory. Due the use of standard Java, we deal with a standard garbage collector that does not have real-time constraints. Our strategy to avoid garbage collection delays is to avoid garbage collection cycles by minimizing the allocation of new objects by the control code. Because the use of third party libraries, not all allocation could be avoided. Therefore, we allocated a large pool of memory, avoiding the requirement of garbage collection during a 30-minute run. Jitter due to context switching is avoided by setting tuning the affinity of our control threads. Of the 16 cores in our control computer, we remove two from the Linux kernel scheduler and then manually schedule our control threads on these cores. During testing, we found that communication between threads on two physical CPU's in a dual socket system introduced jitter in the millisecond range. Therefore, we disabled the second CPU. The combination of minimal object allocation and setting the thread affinity reduced the jitter of our control loops to a minimum and after a short initial warm-up phase, no deadlines are missed during our half hour run.

Another large infrastructure project mainly finished during the previous quarter was the logging computer. We improved upon that and made the system more reliable. A smarter seeking algorithm based on a binary search algorithm made it much quicker to seek through large (more than half an hour) log files, a situation not seen before full integration testing. Another improvement was adding a crop feature for our log data, so that small interesting section can easily be exported and copied to the individual developer's computer for more in-depth analysis of the data.

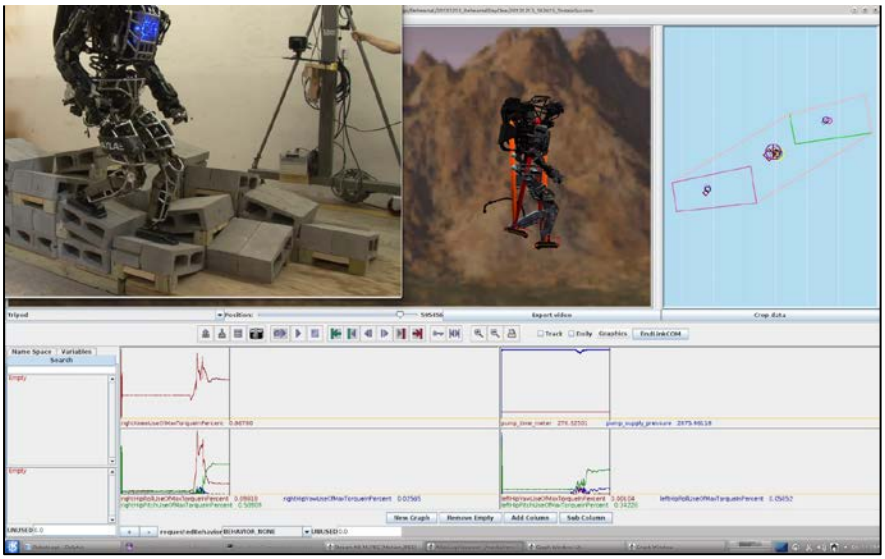

Figure 35. ***Logging computer showing recorded video synchronized with Simulation Construction Set (SCS) playback of data and graphics.***

During the weeks leading up the DRC trials, we made sure that we had a replacement for all our computer equipment, including the field computers, network switches and fiber equipment. We rotated our field computers between the main and backup system daily, so that we had a guarantee that both were up to date and fully functional.

3.2.14 Developer Infrastructure

We maintain and continue to improve upon our developer infrastructure as a way to automate many of the tasks that can easily be put off by a human but that can also lead to catastrophic problems in a project such as this. By automating things like quality assurance, software testing, software integration, complexity management, etc. we can not only decrease the number of bugs and errors introduced to the system but improve the productivity of our developers by allowing them to better focus their efforts and identify problem areas. This was aided in a great part by our reliance on the Atlassian tool suite, which we've been using for years. This was further enhanced by our entering in to a partnership with Atlassian, who contributed the time of one of their engineers as an embedded member of the DRC team contributing both to our software as a regular team member as well as an Atlassian evangelist to help us better utilize the tools. These tools are:

- **Subversion:** Version control. We host our own Subversion server.
- **Atlassian Bamboo:** Continuous Integration server (automated build server, automated test runner, code coverage and test reporting, regression testing, etc.)
- **Atlassian Confluence:** Document and collaboration software
- **Atlassian JIRA:** Issue tracking and Agile project management software
- **Atlassian Fisheye and Crucible:** Source Control Management and Code Review
- **Atlassian HipChat:** Team messaging and chat room software
- **Atlassian Crowd:** Authentication management

Some features we implemented were a digital "Wall Board" that provided a stream of information about the state of the code-base, automating the process of blocking commits unless the commit was being issued to fix the broken build, manual code freeze option, and automatic nightly builds with videos (Fig. 36).

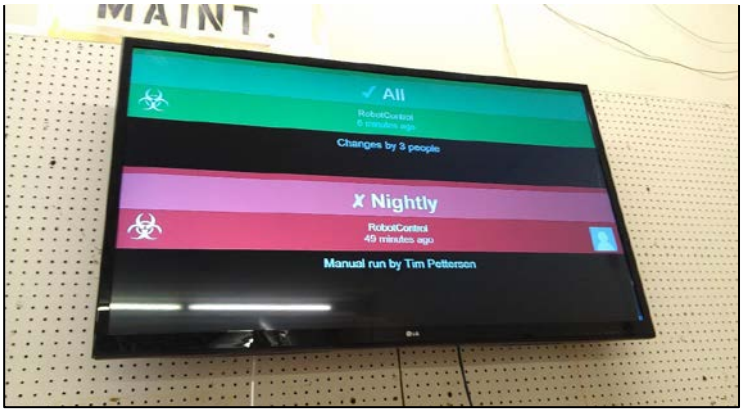

Figure 36. ***Bamboo Wall Board displaying the status of our suite of test cases.***

3.2.15 Gazebo

Gazebo presented significant challenges during the last quarter. The main reason for using Gazebo is for use of the collision modeling which is better than collision handling in our own simulation environment: Simulation Construction Set (SCS). However, for Gazebo t o be of worth, it needed to be stable, and runnable with the iRobot hands that we were planning on using for the competition in December. Since the iRobot hands were never released on the standard DRCSim release, they required special builds of multiple branches of Gazebo and DRCSim. Difficulties in getting the iRobot hands to function in Gazebo had to be repeated as the iRobot hand branches and other dependencies changed multiple times requiring re-working of the process to get Gazebo and DRCSim to be built originally from source and later mixing builds with some nightly builds instead of the original means of getting the Atlas model to load in Gazebo with the iRobot hands. Once the process had been refined, controllers were successfully implemented to control the Gazebo iRobot hands. At this point although our code could control Atlas in Gazebo without the iRobot hands, as soon as the iRobot hands model was loaded new instabilities in the model occurred that were never fully reduced sufficiently for Gazebo to be practically used for practicing any tasks in Gazebo. All tasks were created/loaded for Gazebo, and at multiple times was stable enough for testing some walking related tasks, but never any manipulation.

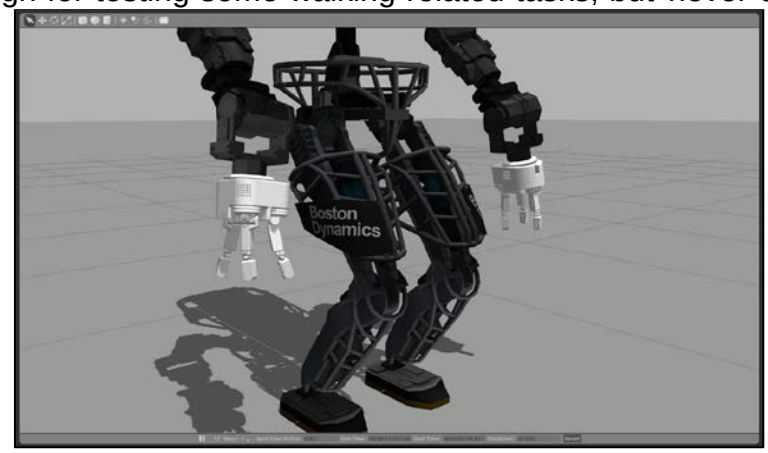

Figure 37. ***Atlas with iRobot hands in Gazebo. Notice how the fingertips are turned the wrong way due to incorrect modeling of the proximal finger joints.***

3.2.16 Qualifying

When it became clear that our controller would not be ready by the Quals deadline, we dedicated a separate team to the task of connecting our UI to the controller that Boston Dynamics developed and supplied with each of the Atlas robots. There were also some problems when matching the intended functionality of our UI with the behaviors exposed by BDI's Application Program Interface (API). We were able to resolve these issues working with BDI and were able to qualify using their provided code and our interface.

Connecting our UI to BDI's API also gave us a useful debugging tool for checking our software bugs. If we began to see unusual behavior while using our controller, we

could shut down, restart using BDI's controller, and check to see if the same behavior was present. If not, then there was probably a software bug on our side; if so, that pointed to a possible hardware problem on the robot and we could bring it to the attention of BDI.

3.2.17 Trials Emulation

In order to develop our system and practice for the competition, an obstacle course was built in accordance with specifications provided by DARPA. Each element of the tasks are placed or secured to a platform, pallet, or freestanding wall to provide a simplistic, portable, and interchangeable layout. We had examples of all tasks except the driving task, which we decided early on would not be possible in the time allotted. We add some variability into each task set up, to ensure we were robust to some variation.

Figure 38. ***Task emulations at the IHMC Robot Lab.***

During the DRC Trials, we would have a single fiber connection from the robot to the field computer. Our initial control setup used two field computers, each connected with a separate fiber to the blue box. A solution was found in procuring a 10GBits Ethernet switch with two SFP+ ports and 10 10GBase-T Ethernet ports. This allowed us to connect both field computers as well as the logging and development computer directly to the robot. To mimic the trials, we acquired the specified media conversion boxes and used this during testing. We saw no problems with this setup in our testing. Unfortunately, during the trials the required network devices changed and required additional setup and testing.

3.2.18 Practicing Competition Tasks

We had example task for every task with the exception of driving. Driving was sufficiently different from all other tasks. The complexity and cost motivated us eliminate this task from our plan right from the start. We knew we would have very limited time and the driving task would require a lot of attention. We were able to demonstrate all the other tasks in our lab prior to the competition, with the exception of the ladder. Though we had a ladder set up for testing, we ran out of time.

3.2.19 Trials Competition in Miami, Florida

Our team did well at the competition. We finished in second place and won “best of” awards for 2 of the 8 tasks.

Figure 39. ***TEAM IHMC at the DRC Trials in Miami, Florida.***

We anticipated scoring a full four points on the wall task, the door task, the valve task and the terrain task. We anticipated one point on the debris task, only because we were a bit slow in execution. We could complete the task, just not in the 30 minutes. We anticipated a two on hose, because threading the hose onto the spigot was very difficult. We completed it once in our lab, but failed numerous times. The competition went very much as expected with minor exceptions. We fell during the valve task. We were aware that our knees are at their torque limits when squatting to turn the valves and we knew we had a bad right knee that was prone to failure. However, we forgot to stand up before walking and so our knee collapsed and we fell. The hose task was a pleasant surprise in that we were granted four points for attaching the hose. We had never attempted the ladder, so we were not confident that we would be able to get any points. With no prior practice, we ended up failing to get any points on the ladder task.

Figure 40. ***Team IHMC’s Atlas performing the task at the DRC Trials.***

3.3 Phase 3: DARPA Robotics Challenge – Finals

Our main accomplishments this quarter were to prepare for the DARPA Robotics Challenge Finals in Pomona, California and compete in the finals. Our main goal was to successfully complete the eight challenge tasks in under 60 minutes. We were able to achieve this goal on our day two attempt, completing the course in 50 minutes and 26 seconds. With this performance, we won second place and were awarded a $1M prize from DARPA.

DRC FINALS TEAM STANDINGS		
TEAM	SCORE	TIME
TEAM KAIST	8	44:28
TEAM IHMC ROBOTICS	8	50:26
TARTAN RESCUE	8	55:15
TEAM NIMBRO RESCUE	7	34:00
TEAM ROBOSIMIAN	7	47:59
TEAM MIT	7	50:25
TEAM WPI-CMU	7	56:06
TEAM DRC-HUBO AT UNLV	6	57:41
TEAM TRAC LABS	5	49:00
TEAM AIST-NEDO	5	52:30
TEAM NEDO-JSK	4	58:39
TEAM SNU	4	59:33
TEAM THOR	3	27:47
TEAM HRP2-TOKYO	3	30:06
TEAM ROBOTIS	3	30:23
TEAM VIGIR	3	48:49
TEAM WALK-MAN	2	36:35
TEAM TROOPER	2	42:32
TEAM HECTOR	1	02:44
TEAM VALOR	0	00:00
TEAM AERO	0	00:00
TEAM GRIT	0	00:00
TEAM HKU	0	00:00

Figure 41. ***DRC Finals Team Standings. Team IHMC completed all 8 tasks in 50 minutes and 26 seconds, which was good for a second-place finish and a $1M prize from DARPA.***

Unfortunately, on our day one attempt we fell while going over the cinder blocks and while climbing the stairs. We were able to do a reset after the cinder blocks and ended up with seven out of eight points. However, the two falls causes some structural shifting of the chest, which threw off calibration on the robot. Because of the calibration misalignment and the fact that we only had a seven point run on the first day, on the second day we decided to use the "slow and steady" approach, sacrificing speed over reliability. This approach worked well in the end and was good for second place. However, our time of 50:26 was much slower than we were capable of. In the laboratory, we were able to complete all the tasks except driving in under 40 minutes. We believe if we were able

to have two more runs at the DRC finals, that on one of them we would have been able to achieve about 35 minutes.

Figure 42. ***Team IHMC fell two times during the first day of the DRC finals. The fall on cinder blocks was due to taking an aggressive step without selecting a walking configuration used for aggressive steps. The fall on the stairs was due to a combination of rushing since we were low on time, a programming bug, and body oscillations that we think were a result of falling on the cinder blocks. After day one, we made corrections for both falls and were able to successfully complete all eight tasks on day two.***

We had a pretty high success rate (better than 4 out of 5 attempts) for all of the tasks in both the laboratory and during practice at the DRC. Our main cause for the fall on the cinder blocks on day one was due to taking too aggressive of a step from the high flat cinder blocks in the middle down to the tilted cinder blocks. Stepping down and forward has always been difficult for Atlas due to the ankle pitch limits. When stepping forward and down the trailing leg can hit these limits, which then cause the CoP to move to the front edge of the foot, resulting in loss of control of the CoP. To get around this limitation, we change a few parameters of the walking commands, including leaning forward so that the robot shifts backwards, thereby reducing the ankle pitch angle, and also increasing the height of the robot. During our cinder block fall, we did not make these adjustments since we did not realize that the step down required them. During the step down, the ankle limit was indeed hit and the robot fell backwards. After resetting after the fall on the cinder blocks, we quickly made it back through the door and to the cinder blocks. We made it over the cinder blocks using the adjustments mentioned above. At the stairs, we were running out of time and the robot was behaving sporadically and we fell on the first step. Upon review of the log file we also discovered a bug in the stair climbing code. We fixed that bug, revised our stair climbing strategy, and did some practice runs such that on day two stair climbing went pretty smoothly.

Figure 43. ***Team IHMC Running Man completing the 8 tasks, plus celebrating at the end, at the DARPA Robotics Challenge Finals.***

Figure 44. ***One of our goals for the DRC was to engage the crowd and allow visitors to see the robot and operator interface. We had three IHMC "Blue Men" release nearly 200 T-shirts into the crowd during our two runs. In our garage, we gave demonstrations to over 100 people and interviews with over 10 press organizations.***

3.3.1 Car Driving

The Atlas robot was too big to fit into the driver's seat of the DRC vehicle. Our approach, as well as those of the other Atlas teams, was to have the robot drive the vehicle from the passenger seat. We made two vehicle modifications to enable Atlas to steer and press the gas pedal. The steering modification (Fig. 45) was a handle that clamped to the steering wheel. The handle was connected to the mount via a spring, which provided compliance between the robot hand and the steering wheel, yet still transmitted tangential forces into steering torques. We created a passenger side gas pedal (Fig. 45) by using a hydraulic cylinder and a fluid transmission. Pressing on the passenger side cylinder caused the driver's side cylinder to extend and press the vehicle's gas pedal.

Figure 45. ***Left: picture of the vehicle modification for steering. Right: Image of vehicle modification for pressing the gas pedal.***

To facilitate localization of the robot in the vehicle, we mounted a checkerboard fiducial marker (Fig. 46) onto the hood of the vehicle. We determined the transform from the checkerboard pattern to the vehicle by having a person align the vehicle model to the

LIDAR scan points. From then on, we determined the transform of the robot to the vehicle processing the optical image of the checkerboard pattern and determining its transform to the camera system.

Figure 46. ***Picture of the vehicle with the checkerboard sticker mounded on the hood. The checkerboard acted as a fiducial marker to enable automated localization of the robot in the vehicle.***

3.3.2 Car Egress

Due to the joint limits of Atlas, the robot was not able to step down the full height of the vehicle in a single step. Therefore, the vehicle was modified with a step on the passenger side. The step provided an intermediate platform so that the robot did not have to step down the full vehicle height in a single step.

Figure 47. ***Picture of the vehicle with the step addition. The step provided an intermediate platform so that the robot did not have to step down the full vehicle height in a single step.***

The first phase of the car egress is performed using stiff position control on each of the joints; a simple script was created using the infrastructure provided by the Whole Body IK module, where the coordinated motion of each of the limbs can be easily designed in task space.

As soon as the robot stands up inside of the vehicle, the joint controller is switched from position to force and the walking controller is activated; the second phase of the egress is therefore nothing more than “walking” outside of the vehicle.

3.3.3 Practice, Practice, Practice

We spent the last couple weeks before the finals practicing several times a day. With this method, the pilots could train in conditions close to the competition and at the same time highlight weaknesses and bug in our codebase. It also allowed to check in which of the tasks we could win some time, and develop the features necessary to decrease our overall time to accomplish the tasks.

3.3.4 Robot Operation

Our operator interface required a primary operator, co-pilot, and two high-level planners and strategists. Only the primary operator would interface directly with the robot. The co-pilot maintained a checklist crafted from the mistakes and required reminders of previous practice runs. The planners and strategists would construct the high-level goals and order of the run as the co-pilot would call out technical reminders and check steps for the eight tasks in real time. One of the mission planners would hold a radio that the field team would use to confirm points achieved.

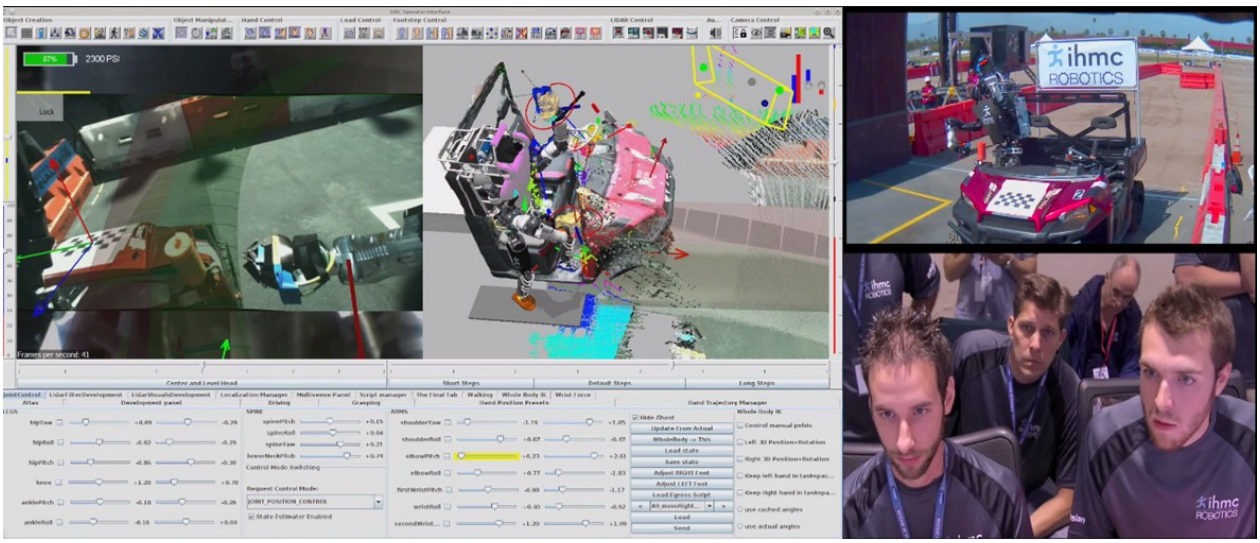

Figure 48. ***Operator UI (left), Primary operator (John Carff), Strategist/Planner (Matt Johnson), Co-Pilot (Duncan Calvert), Strategist/Planner/Field Team Communicator (Jerry Pratt’s arm in top right).***

3.4 Phase 4: Extended Research

For this final extended year of the DRC project, we focused on a) increasing the amount of autonomy in humanoid behaviors, while allowing for coactive design of user interfaces, b) walking on surfaces with only partial footholds, such as on the tops of pointy rocks, c) Using upper body angular momentum to help balance, and d) improving the software architecture and interface to the whole-body momentum based control framework.

Our main goals were to develop walking algorithms for more dynamic maneuvers and to achieve mobility through more complex 3D environments. We made significant progress on these goals, but still have quite a way to go in order to achieve mobility on par with humans. In related ongoing and future projects, we will continue towards these goals, and also start to investigate more application scenarios, such as assembling modular space structures, additional disaster recovery scenarios, and power plant inspection.

The following results were all achieved in a lab setting under ideal, controlled conditions. Namely, optimal lighting, flat walking surfaces, clear and well defined walking paths, and no degradation or blackouts in network communications. Our procedure for operating Atlas and developing new functionality is three-pronged, with effort divided between logging, testing and debugging, and running on physical hardware.

3.4.1 Logging

One of our strongest software assets when dealing with the physical Atlas is our logging suite. The logging component is an extension of our simulation environment, in which the physical robot state is fed into the visual interface along with all controller parameters and inputs. The output from the physical Atlas's state estimator is used to drive the joints of the computer graphical model Atlas in real time. All of this information is timestamped, logged, and merged with video from two dedicated high definition lab cameras focused on Atlas. Analyzing this data allowed our team to identify whether failures were caused by software bugs, hardware problems, misconceptions about the robot, or missed edge cases. Once the issues are detected from the logger we document and push the issue to simulation replication and testing.

3.4.2 Testing and Debugging

With the addition of the physical component, an all-encompassing testing approach was preferred, though impractical due to bottlenecks in physical running constraints. A tiered approach, using our existing simulation unit tests combined with black-box testing of the physical robot helped eliminate weaknesses in our software. Figure 1 shows our approach to black-box walking behavior failures, simulation refinement, and improved test coverage. The simulation recreation fork allows us to investigate limitations of the simulation and sources of error from the robot. In instances where the simulation was unable to recreate the failure, we adapted the simulation to coincide with the real world as closely as possible, placing a higher trust in the simulated environment, and reducing the need for physical confirmation of successful operation. Once the failure was introduced, a simulation test is created based on these conditions and the controller is refined until the test passes along with the regression test suite. The final changes to the controller are validated using the physical Atlas on the next round of black-box testing.

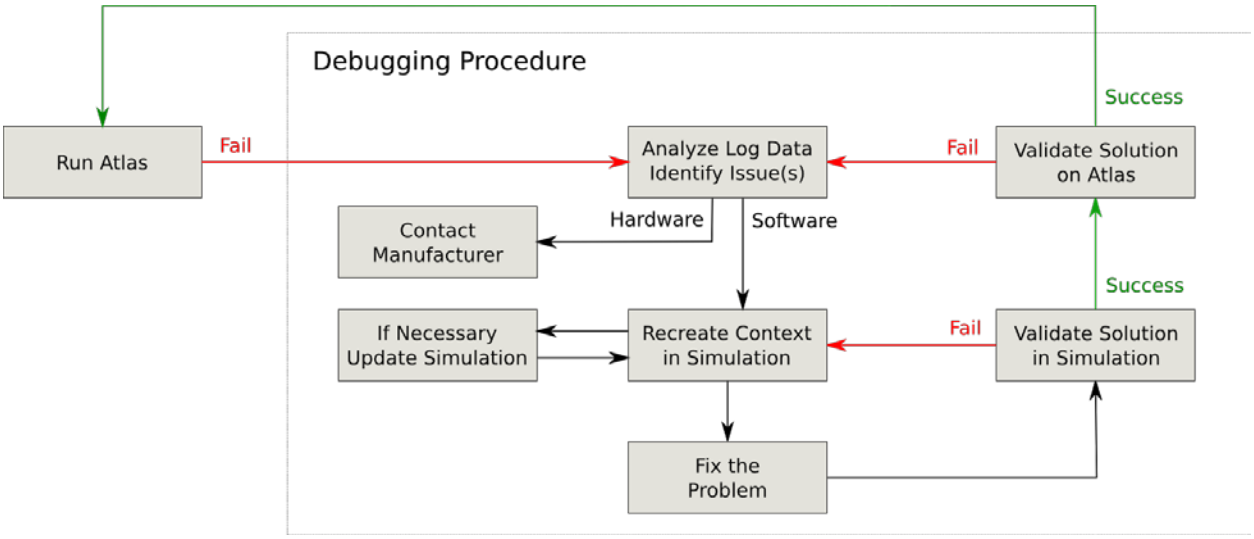

Figure 49. ***Procedure chosen to improve the controller reliability.***

3.4.3 Running on Hardware

Only after new features were fully vetted in simulation and our regression test suites were they attempted on actual robot hardware. This reduced possible downtime resulting from incomplete feature implementations or buggy software. When testing a new feature on hardware, the implementer works with a robot operator to first test the software in its main use cases. If everything works as expected, the robot operator is then responsible for challenging the implementer with edge cases that may cause the new functionality to fail or behave in unintended ways. The feature isn't considered complete until it performs in an intuitive way in all use cases. In some cases, if a particular feature needs to be very robust (e.g. planning a path over uneven terrain), an untrained robot operator will attempt to perform a task that exercises this functionality. This is useful because it is likely that an untrained or inexperienced operator will try to complete the task in a different manner than someone who has learned the optimal techniques and strategies for interfacing with the robot.

3.4.4 Enhancing the IHMC Control Framework

With the need of enabling fast development, better support of the controller, and removing numerous control limitations due to a bad infrastructure, we decided to rewrite a significant portion of the IHMC controller.
An initial set of goals was defined as:

- Review the controller API and the Java/ROS packets to reach a stable definition of the API.
- Extract the core of the controller: the feedback controllers, the QP solver, and the inverse dynamics algorithm. This new control module's API will be thread safe to be able to benefit from a multi-threaded infrastructure.

- Redesign the trajectories used to control any part of the body to able to execute complex trajectories by using waypoints. By doing so, the controller will be able to execute any type of trajectory planned by a third party module.
- All the control elements related to the robot's balance that are scattered in the controller should be reassembled into a single balance control module. This will allow a drastic improvement in terms of visibility, preventing bugs.
- Redesign the main control class that was impractical, unreadable, and was preventing any new employee to catch up quickly.
- Migrate to use a new QP solver to improve control authority flexibility.
- Review the multithread infrastructure.

As the redesign of the controller would not produce direct results, we limited the duration of the redesign to 2 months. Among the initial goals, only the last one was not completed.

During this redesign, additional improvements were achieved:

- A new infrastructure for the controller core that supports three distinct control frameworks: whole-body inverse kinematics, whole-body inverse dynamics, and whole-body virtual model control. This new controller core is already used is different controllers and is beneficial in speeding up the design and implementation of new controllers.
- Hybrid force-position control when using the whole-body inverse dynamics framework. Even with the presence of position controlled joints on a force controlled robot, the control framework remains unified. This used to be a flaw in the previous controller often source of bugs.

Direct benefits of the redesign include:

- Direct control of the center of pressure of each foot: used for foothold exploration, see next section.
- Direct control of the overall rate of change of angular momentum: used to extend balance control when walking over partial footholds, see next section. Also used to counter the angular momentum generated by the swing leg.
- Development on the controller has been sped up.

3.4.5 Walking Partial Footholds

As an effort to improve the balancing capabilities and the robustness of our walking algorithm we have addressed the problem of stepping and walking on small footholds. Ultimately this will help making legged robots useful in real world scenarios, where the environment is not always flat and can support the entire area of the foot. We have focused our experiments on situations where the exact geometry of the surroundings is unknown. In the course of the past year we have improved our walking algorithm to automatically detect partial footholds and balance on them. After each step the robot shifts its weight around in the foot to check if the it is fully supported by the ground. If this is not the case the part of the foothold that was not able to support weight is removed. In the case of small footholds such as line contacts, the walking algorithm automatically adjusts to the smaller support area. We have shown the Atlas humanoid walking on partial footholds in simulation (Fig. 50) and on the real hardware (Fig. 51) using this strategy.

To achieve balancing on such small footholds we adjusted the control algorithm to use upper body angular momentum for balancing. The controller redesign during the last year enabled us to produce these whole body balancing motions in a clean and intuitive way. Usually angular accelerations of the upper body are kept small and the posture of the robot is controlled to be upright. By temporarily disabling this objective, human like whole body balancing motions are generated.

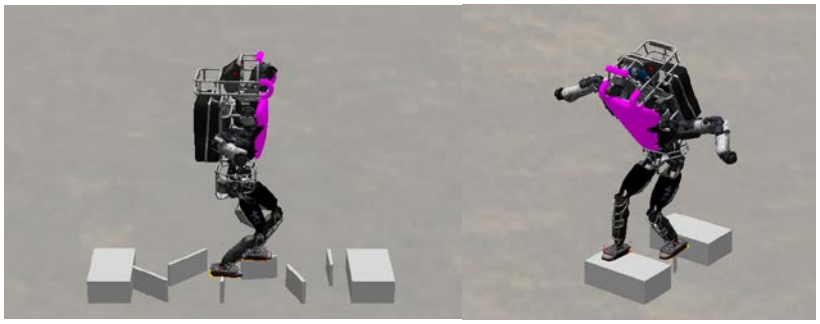

Figure 50. **The simulated Atlas robot walking over randomly oriented, line shaped stepping stones and a point foothold. After each step the new foothold is explored and the control algorithm adjusts the stepping accordingly.**

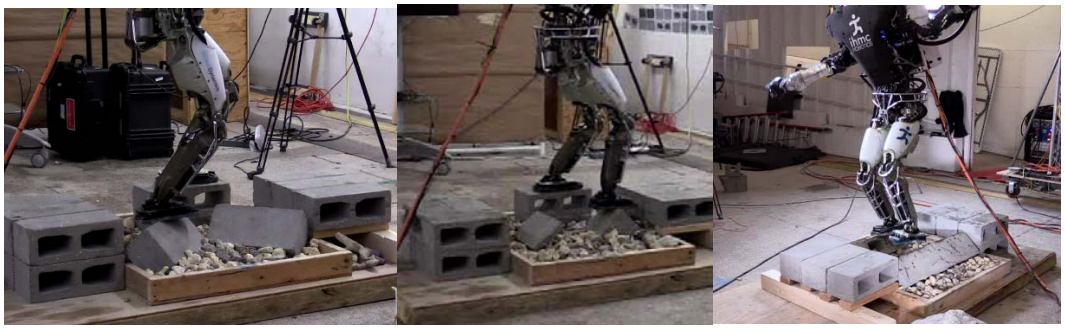

Figure 51. **The Atlas humanoid walking over variously oriented line contacts. We use tilted cinder blocks to produce line contacts. It can be seen that the feet of the robot stay flat and do not conform to the cinder block surfaces.**

3.4.6 Autonomy of High Level Behaviors While Retaining Coactive Design Methodology

In our effort to increase robot autonomy and operation speed, we have developed a toolset for implementing high level behaviors into our controllers. High level behaviors consist of a series of simple behaviors that break down a task into simple pieces. Each simple piece is a building block that can be easily used in a wide variety of high level behaviors in combination with other simple behaviors. An example of a high-level behavior is our “Pick Up Balls” behavior (Fig. 51). This behavior consists of several simple sub-behaviors, such as turn on LIDAR, walk to a location, and grasp. Some of these simple tasks were created specifically for this behavior and some are reused from other behaviors. Figure 52 shows the flowchart of the ball pickup behavior, and all of the simple behaviors that it is comprised of. As our developers implement behaviors, our simple behavior set increases, making all future high level behaviors easier and quicker to develop and implement.

All behaviors have a coactive layer built in. This allows the developer to easily send relevant information to the user interface that will inform the operator of the robot’s behavior state, current intent, and any other information that the developer believes will aid in successful operation. This coactive layer allows bidirectional communication; the behavior to informs the operator, and allows the operator to augment the behavior in real time as necessary. For example, during a high-level behavior, the robot begins to walk and the operator sees that one of the automatically generated footsteps is misplaced. The user can then pause the behavior, correct the footstep, and then resume where the behavior left off without having to restart the current operation. This coactive element, built into our behavior toolset, greatly increases the operator's trust and confidence in the system, increasing speed and reliability during operation.

Our behavior toolset greatly reduces operator workload and burden by moving the majority of the tasks onto the behavior side instead of relying solely on the operator. This is not to remove the operator from the equation but to increase operator reliability, speed, and capabilities be giving the user a tool that can be used to offload simple, and repetitive tasks. Not only does the operator benefit from the autonomy of the behaviors, but our coactive design also allows the behaviors to benefit from the operator capabilities in situations where it may be slow or unreliable. For example, if a behavior is attempting to find a specified object in a cluttered environment that the operator can already see, the operator may reduce the search area or simply tell the behavior where the object is. This allows the behavior to utilize the operator's ability for recognition, while the operator can be less burdened with tasks that the behavior is good at like how to grasp an object in three dimensional space.

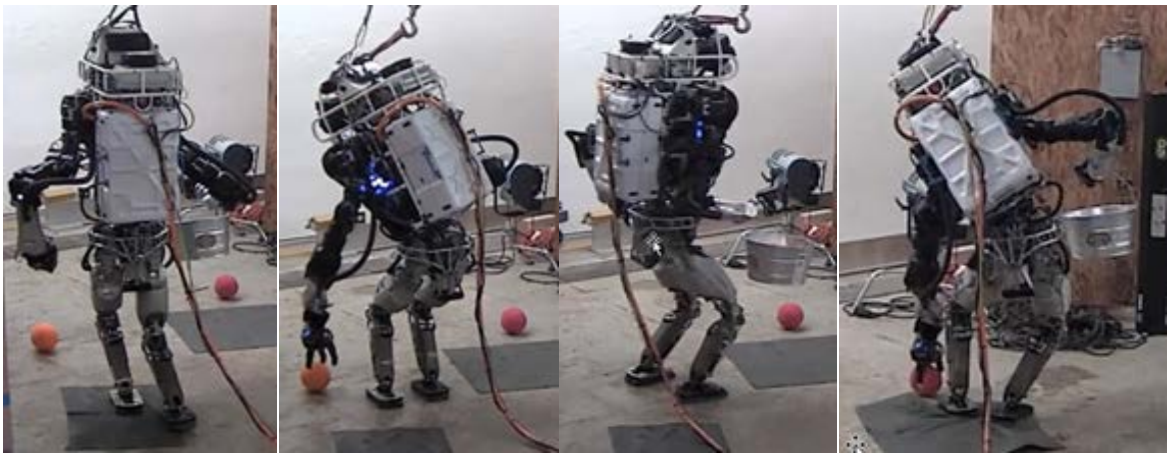

Figure 52. **Atlas autonomously picking balls off of the floor using a high level behavior. While this is a very simple behavior, it demonstrates a fully autonomous task that integrates sensing, perception, whole body mobility, and simple manipulation.**

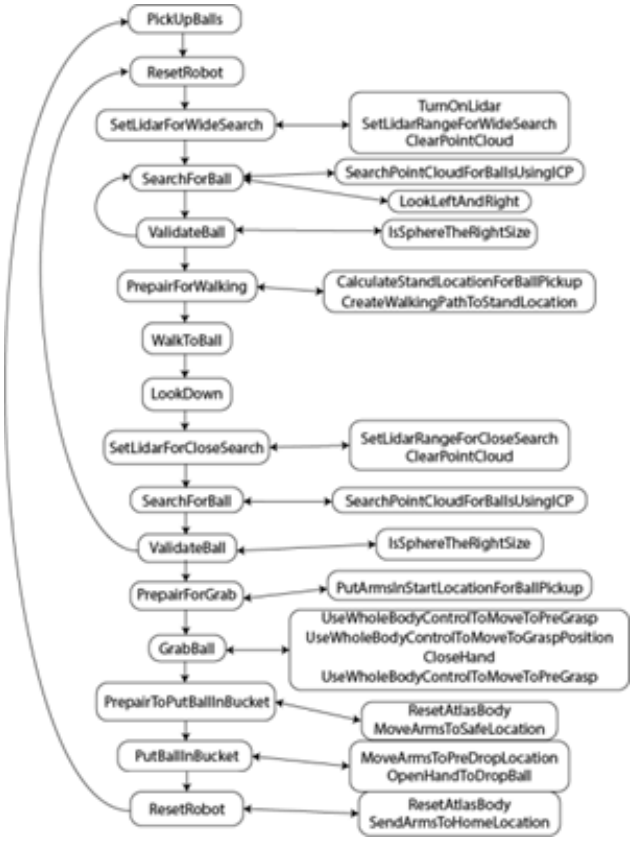

Figure 53. **Behavior flow chart for autonomously picking up balls with Atlas.**

4.0 RESULTS AND DISCUSSION

This section gives a summary of the results of each phase of the DRC. Due to the structure of this report we present a list of results from each section and highlight our findings.

4.1 The VRC: Results

This part of the competition was designed so teams could develop their robot capabilities in a simulation environment. At the same time restrictions on the communication between operator and robot encouraged the teams to use autonomous behaviors as well as optimize the information flowing to and from the robot.

- IHMC developed their controller capabilities further and in this stage we set the foundation for a sophisticated control framework built to control and balance generic legged robots. The architecture of the controller was specific to the GFE robot, but it had all the main components that we later extracted to create a robot agnostic controller that is now used on multiple robots worldwide including the Valkyrie robot.
- Due to the bandwidth limitations we investigated options to limit the flow of information to and from the robot. We developed an optimized interface that would present the operator with precisely the information necessary to accomplish his mission. We found that by doing this we were able to keep the autonomy of the robot minimal and allow the operator to efficietly control the actions of the robot.
- Tools were developed to aid with manipulation in a simulated environment. The balancing controller was extended to cope with environment interactions through the hands and other parts of the robot.
- All software tools were developed such that a failure in one tool would not cause to robot to fail. Parts of the software can be restarted without problems. This was a major reason for our success in the DRC. It is important to anticipate failure rather then to assume it can be avoided altogether.
- We spent time integrating development tools by Atlassian into our work flow. This ultimately helped to make development more efficient and less prone to errors. It also allows organizing work effort and focusing on important tasks.

4.2 The Trials: Results

The development effort of this phase was focused on moving the control software that was developed in the previous state the the real Atlas robot. The difficulties when moving to a real world robot are often underestimated. Sensor noise, manufacturer tolerances, and actuation imprecisions are just the most commonly named ones. Testing new software on a real platform introduces the risk of breaking hardware components which sets back development. This is especially critical in a competition were time is a limited resource. We found that the Atlas robot build by BDI was extremely robust and repeatable which is not a common feature of robotic systems.

See publications (Johnson - Team IHMC's lessons learned from the DARPA robotics challenge trials) for a more in depth analysis of the Trials.

- The team improved the robustness of the balancing controller to cope with sensor noise and actuator imprecisions. This was an important step in creating a robot control software that can handle a variety of actuators and senors and it helped us understand the limitations of our software and improve it.
- The user interface was extended to include an augmented reality that could be extended by the operator placing modifiable objects in the world. This enables more efficient robot control and reduces the bandwidth needed to present the operator with enough information to accomplish the mission.
- To debug problems on the robot we developed a logging system. This system records the robot state as well as controller variables. These variables can be played back, synchronized with a video of the robot. We have extracted this tool and made it robot agnostic such that it can be used with any robotic system. It enables us to efficiently debug and improve the controller software.
- Manipulation is a very challenging problem on a real robot. The main reason is the flexibility that we expect from robotic hands also makes it fragile. After breaking a lot of hands we adjusted our strategy to include a hand and an alternative hook.

4.3 The DRC Finals: Results

The final stage of the competition was a one hour mission that included driving a vehicle, traversing difficult terrain, and manipulating objects. To achieve this the robustness of the control algorithm needed to be improved greatly and switching between different control modes had to be possible (e.g. 'driving in a stated position' to 'walking and balancing'). The bandwidth limitations were increased. Due to that and the time limit of the mission more autonomy was needed and we integrated tools including computer vision into the software stack. The robot fell twice during this stage. In one case the reason was operator error, in the other case it was caused by a software mistake.

See publications (Johnson - Team IHMC’s Lessons Learned from the DARPA Robotics Challenge: Finding Data in the Rubble) for a more in depth analysis of the DRC finals.

- During the final runs of the competition we employed a co-pilot operator. This second operator was in charge of maintaining checklists and to observe and intervene of the main operator was about to make a mistake. We collected data on this interaction and showed that a lot of failures can be avoided this way.
- The time and bandwidth limits caused us to integrate helper tools based on computer vision and a microphone sensor into the robot software. This allowed a more efficient use of the bandwidth and saved time by helping the operator locate objects in the world.
- As part of the preparation we employed a very strict training schedule for our operators. This helped identify possible complications as well as train them to navigate the challenge quickly and efficiently.

4.4 Extended research: Results

During the challenge, it became obvious that robotic systems and not yet ready to be employed in disaster scenarios like intended. They lack balancing capabilities, autonomy, and the ability to recognize problems and react to them. During the challenge itself these issues could be overcome by optimizing the operator interaction with the robot. After the challenge team IHMC spent much effort into investigating some of the aspects that would enable a robot to do useful work in the real world.

- The control algorithm was extracted and improved such that it is capable of running generic legged robots. It is currently employed on the Valkyrie robot by NASA to help teams compete in their Space Robotics Challenge.
- As part of the effort to make humanoid robots more robust the balancing algorithm is extended to allow walking on limited footholds and use human like recovery motions with the upper body. We will continue to work on more robust walking behaviors since we believe robust locomotion is crucial to make these systems usable in everyday life.
- To improve the autonomy of robotics systems we are developing a behavior framework that allows the robot to accomplish tasks with less operator input. At the same time the operator maintains full control over the robot having insight into the next actions of the robot and the ability to intervene.

5.0 CONCLUSIONS

Through the DRC, Team IHMC made significant improvements in the state of the art in the areas of bipedal walking algorithms and coactive user interfaces that support robot-human teams. Our software development methodology resulted in high quality, efficient, reusable software. Much of this software is now being used by several research groups outside of IHMC. We were successful in all phases of the competition, placing first in the VRC, second in the DRC Trials, and second in the DRC Finals.

Over the past year we have continued to adapt and add new features in these areas while maintaining robustness. Our walking and balance algorithm is now capable of handling pointed surfaces and line contacts, two challenges not seen in the DRC. We have also added new levels of autonomy to our Atlas behaviors, allowing human operators to pass more task responsibility to the robot while retaining the power to intervene and perform error correcting measures. We believe these are all essential building blocks for moving forward with the goal of achieving higher level autonomy for bipedal robots.

There is still much more work to be done in making all of these components more robust though. We would first like to add to our library of autonomous behaviors and develop a way to dynamically switch between them, with the ultimate goal of completing the DRC tasks completely autonomously with a relatively high reliability rate. For this we will likely need to delve into areas of research which is new to us, including computer vision, machine learning, and highly distributed systems.

Because of the DRC, and because of work performed by IHMC, humanoid robots are now significantly closer to being useful in real world scenarios. There is still much work to be done, but the DRC has laid a firm foundation. Going forward IHMC is continuing to lead the charge for getting these robots out of the lab and into the field.

6.0 PUBLICATIONS AND PRESENTATIONS

6.1 Publications

Johnson, M., Shrewsbury, B., Bertrand, S., Calvert, D., Wu, T., Duran, D., … Pratt, J. (2016). “Team IHMC’s Lessons Learned from the DARPA Robotics Challenge: Finding Data in the Rubble.” *Journal of Field Robotics*.

Johnson, Matthew, et al. "Team IHMC's lessons learned from the DARPA robotics challenge trials." *Journal of Field Robotics.* **32.2**, pp.192-208, 2015.

Johnson, M. (2014). Coactive Design: Designing Support for Interdependence in Human-Robot Teamwork. PhD Thesis - Delft University of Technology.

Johnson, M., Bradshaw, J. M., Hoffman, R. R., Feltovich, P. J., & Woods, D. D. (2014). “Seven Cardinal Virtues for Human-Machine Teamwork: Examples from the DARPA Robotic Challenge.” *IEEE Intelligent System*s, Vol. **29** No. 6, pp. 74-80, November 2014.

Johnson, M., J.M. Bradshaw, P. J. Feltovich, C. M. Jonker, M. B. van Riemsdijk, and M. Sierhuis. “Coactive design: Designing support for interdependence in joint activity.” *Journal of Human-Robot Interaction*, Vol. **3,** No. 1, pp. 43-69, 2014.

Johnson, Matthew, J.M. Bradshaw, Paul J. Feltovich, Robert R. Hoffman, Catholijn Jonker, Birna van Riemsdijk, and Maarten Sierhuis. Beyond Cooperative Robotics: The Central Role of Interdependence in Coactive Design. IEEE Intelligent Systems, May/June 2011 (vol. 26 iss. 3), pp. 81–88. Reprinted in Hoffman, Robert R. (author, editor), J. M. Bradshaw, Pat Hayes, and Kenneth M. Ford (editors). Collected Essays on Human-Centered Computing, 2001–2011. New York City, NY: IEEE Press, 2012, pp. 95–102.

Johnson, M., Bradshaw, J. M., Feltovich, P. J., Jonker, C., van Riemsdijk, B., & Sierhuis, M. (2012). “Autonomy and interdependence in human-agent-robot teams.” Intelligent Systems, IEEE, Vol. **27**, No. 2), pp. 43-51, 2012.

Koolen, Twan, et al. "Summary of team IHMC's virtual robotics challenge entry." *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, 2013.

Wiedebach, G., Bertrand, S., Wu, T., Fiorio, L., McCrory, S., Griffin, R., Nori, F., and Pratt, J. “Walking on partial footholds including line contacts with the humanoid robot Atlas.” *2016 IEEE-RAS 16th Annual International Conference on Humanoid Robots (Humanoids).* IEEE, November 2016.

6.2 Presentations

- 15 Nov 2016: “Haptic Terrain Exploration for Biped Robots” – Humanoids 2016, Mexico
- 29 JUL 2016: Workload Assessment workshop: “New Perspectives on Workload in Human/Machine Teams”
- 17 MAY 2016: FLAIRS Keynote – “No AI is an Island” – Key Largo, FL
- 29 FEB 2016: Presented on Humanoids and Robotics at the PCC Engineering Forum
- 20 FEB 2016: Presented “Rise of the Humanoid Robots” at PensaCon
- 2 NOV 2015: Presented about humanoid robots at Border Sessions – the Netherlands
- 15 OCT 2015: Organized and presented Human & Machine Teamwork in the Air Force Context
- 29 AUG 2015: Tutorial on Coactive Design for NPS - IHMC
- 18 MAR 2014: Presented Coactive Design to NASA’s Future In-Space Operations workgroup - telecon
- 03 FEB 2014: Presented Coactive Design to Marine Corps Warfighting Lab and Naval Post Graduate School
- 19 NOV 2014: Presented at Coactive Design to National Robotics Initiative PI meeting – Washington DC
- 02 OCT 2014: Invited Speaker at Cooperation of Robots and Sensor Networks Summer School - Germany
- 10 JUN 2014: Presented at Interactive Intelligence workshop - the Netherlands
- 28 APR 2014: Class lecture for TU Delft Context project – the Netherlands
- 17 APR 2014: Presented DRC work to NASA Ames Intelligent Robotics Group – NASA Ames
- 04 APR 2014: Plenary talk at Tulane Engineering Forum – New Orleans
- 01 APR 2014: Presented DRC to local Rotary club - Pensacola
- 25 MAR 2014: Presented Coactive Design at HRI - Germany
- 27 MAR 2014: Presented DRC work to II and 3ME groups at Delft – The Netherlands
- 31 JAN 2014: Presented DRC work Association for Unmanned Vehicle Systems International - Pensacola
- 13-17 JAN 2014: Organized and ran Coactive Design workshop for DSO National Laboratories - Singapore

7.0 LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

Symbols

J_i	Selection matrix
$\dot{v}$	Joint accelerations
p_i	desired acceleration for i^{th} joint
A	centroidal momentum matrix
B	desired rate of change of centroidal momentum
W_i	external wrenches
Q	contact matrix
ρ	contact point weights

Abbreviations and Acronyms

API – Application Program Interface

AR – Augmented Reality

BDI – Boston Dynamics

CMP – Centroidal Moment Pivot

CoM – Center of Mass

CoP – Center of Pressure

DARPA – Defense Advanced Research Projects Agency

DRC – DARPA Robotics Challenge

GFE – Government Furnished Equipment

GUI – Graphical User Interface

ICP – Instantaneous Capture Point

IHMC – Florida Institute for Human and Machine Cognition

IK – Inverse Kinematics

JME – Java Monkey Engine

JSch – Java Secure Channel

LIDAR – Light Intensity Distance and Ranging

OCU – Operation Control Unit

OSRF – Open Source Robotics Foundation

PD - Proportional Derivative

PPS – Pulse per second

ROS – Robot Operating System

SDF – Simulation Description Files

SSH – Secure Shell

TCP – Transmission Control Protocol

UI – User interface

VRC – Virtual Robotics Challenge

XML – Extensible Markup Language